Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GPS	Global Positioning System
GMH	Anwendungsbeispiel: Guide Me Home
SDK	Android (in diesem Fall) Software Development Kit
JDK	Java Development Kit
MEMS	Mikro-elektro-mechanische Systeme
ADT	Android Development Tool
LFCS	Lead Frame Chip Scale (Package)
APK	Application Package File (Google)
API	Application Programing Interface

1 Übersicht

1.1 Motivation und Zielsetzung

Die ursprüngliche Motivation zu diesem Thema wurde angeregt durch die Idee eine Rückführung zu einem Ausgangspunkt zu ermöglichen, die nicht auf GPS basiert. Als Anwendungsbeispiel war das Wiederfinden eines Fahrzeugs in einer Tiefgarage präsent. Neben Überlegungen hinsichtlich Ortung durch Peilung, ergab sich die Idee einer Lösung auf Basis von Trägheitsnavigation.

Trägheitsnavigation nutzt die auftretenden Beschleunigungen und Drehraten, um daraus die Positionsänderung abzuleiten. Als Hardwareplattform wurde eine Kombination aus gängigen Mikroprozessoren und entsprechenden Beschleunigungssensoren angedacht. Um bei einem überschaubaren Rahmen zu bleiben, wurde eine Variante auf Basis von Android Smartphones in Erwägung gezogen. Warum Android wird weiter unten beschrieben. Smartphones verfügen bereits über entsprechende Sensoren und mit Android auch über eine Entwicklungsplattform, die diese relativ einfach nutzbar machen.

So entstand das Thema Navigationsaufgaben mittels Android zu lösen. Als weitere Zielsetzung wurde festgelegt, dass diese Arbeit insbesondere anderen Studenten dienen soll, einen Zugang zur Nutzung von Sensoren unter Android zu finden.

Android ist derzeit das verbreitetste Betriebssystem für Smartphones. Informationen zu Android findet man u.a. im Internet auf deren Homepage /1/. Android wurde 2003 von Andy Rubin gegründet, von Google 2005 gekauft und in die Open Handset Alliance eingebracht. Google ist Hauptmitglied der Open Handset Alliance. Seit 2008 ist Android als Betriebssystem in dieser Form bekannt und wird laufend weiterentwickelt. Aktuell ist Android in der Version 4.0 verfügbar. Im Gegensatz zu anderen Marktteilnehmern ist Android als Quellcode-offenes Betriebssystem verfügbar. Das bedeutet, selbst der Quellcode für Android und die darin bereits enthaltenen Anwendungen sind, wie bei Linux, der Community zugänglich.

Diese Arbeit widmet sich dem grundsätzlichen Verständnis von Navigationsaufgaben unter Verwendung von Sensoren, wie sie in gängigen Android Smartphones eingebaut sind und der programmtechnischen Nutzung durch Android. Die Arbeit soll Studenten dienen einen Zugang zu der Materie zu finden und Neugierde für die Implementierung von Anwendungsfällen wecken.

1.2 Kapitelübersicht

Die Arbeit beginnt mit dem Stand der Technik zu Android Smartphones und deren wesentlichen technischen Merkmalen. Die enthaltenen Sensoren für das Messen von Beschleunigung und Lage des Gerätes werden am Beispiel eines am Markt erhältlichen Gerätes aufgezeigt.

Weiters werden die Grundlagen zum Thema Trägheitsnavigation behandelt. Hierfür werden die Sensoren im Allgemeinen und deren Funktionsweise auf Basis der physikalischen Gesetze beschrieben.

Das nächste große Kapitel widmet sich der Android Entwicklungsumgebung. Neben einer Einleitung zu Geschichte und Hintergrund von Android, werden die wesentlichen Bestandteile und das Ablaufkonzept beschrieben. Ebenso wird die Ansteuerung der einzelnen Sensoren behandelt und mit Codierungsbeispielen demonstriert.

Anschließend wird ein Anwendungsbeispiel, anhand einer Aufgabenstellung und der angestrebten Funktionsweise, ausführlich beschrieben. Dazu wird ein Algorithmus konstruiert und in weiterer Folge ein funktionsfähiger Prototyp entwickelt und erläutert. Der entwickelte Sourcecode findet sich im Anhang und auf einer angeschlossenen DVD.

Im letzten Kapitel erfolgen eine Bewertung und ein weiterer Ausblick auf das gewählte Thema.

2 Stand der Technik

2.1 Beschreibung von Android Smartphones

Wie in der Einleitung bereits definiert, werden die Geräte, welche im Rahmen dieser Arbeit untersucht werden, als Android Smartphones bezeichnet. Dabei bezeichnet Android das eingesetzt Betriebssystem, welches in einem der folgenden Kapitel genauer beschrieben wird, und Smartphones die Geräte selbst. Wie schon der Name verrät, handelt es sich dabei um Telefone, die mit weiteren, „smarten“ Funktionen ausgestattet sind bzw. über Bauteile verfügen, welche durch das eingesetzte Betriebssystem diese Funktionen bereitstellen.

In der Regel verfügen die Geräte neben den, für ein Telefon relevanten, Bauteilen, einen oder mehrere Mikroprozessoren, einen Hauptspeicher, ein Touchdisplay, eine oder mehrer Kameras, USB Schnittstellen sowie Aktoren und Sensoren. Letztere werden in der vorliegenden Arbeit noch näher beschrieben.

Dem Autor stand während der Ausarbeitung ein Smartphone der Marke Samsung bzw. Google zur Verfügung. Konkret handelt es sich dabei um ein Gerät der Marke „Google Nexus S“, welches von Samsung für Google gebaut wurde und ebenso als Samsung verkauft wurde. Es kam im Jahr 2011 auf den Markt.

Um den Stand der Technik festzuhalten werden die technischen Eckdaten beleuchtet.

Abbildung 1: Google Nexus S (Quelle: Google)

Kenngröße	Daten
Hersteller	Samsung für Google
Betriebssystem	Android 2.3.6 Gingerbread (Nach Update Android 4.0.4 Ice Cream Sandwich)
Speichergröße	16 GB iNAND Flash-Speicher + 512 MB RAM

Prozessor	ARM Cortex-A8 (Hummingbird), 1 GHz mit PowerVR SGX 540 GPU

Tabelle 1: Spezifikation Google Nexus S(Quelle: Google)

Die Details zu den eingebauten Sensoren gehen aus dem Datenblatt nicht hervor. Zur Ermittlung dieser Details wurde die Android Anwendung „Android Hardware Info“ von Sleek Apps installiert. Die dabei erkannten Sensoren sind in Tabelle 2: Sensoren Google Nexus S aufgelistet.

Die Liste wird angeführt von einem Beschleunigungssensor, gefolgt von einem magnetischen Feld Sensor und einem Gyroskop. Mit dem Beschleunigungssensor wird die Beschleunigung in alle 3 orthogonal aufeinander stehenden Richtungen gemessen. Mit dem magnetischen Feld Sensor wird das auf das Gerät wirkende magnetische Feld gemessen. Mit dem Gyroskop werden Drehbewegungen entlang der 3 Achsen gemessen. Funktionsweise und weitere Details werden in den folgenden Kapiteln ausführlich behandelt.

Bei den weiteren Sensoren fällt auf, dass diese von Google hergestellt werden. Dies liegt daran, dass es sich dabei um virtuelle Sensoren handelt, die durch sogenannte „Sensor Fusion“ bereitgestellt werden. Bei Sensor Fusion werden mehrere physische Sensoren – Beschleunigungssensor, magnetischer Feld Sensor und/oder Gyroskop – kombiniert genutzt, um genauere oder besser nutzbare Sensorfunktionen zu erzielen. Aus dem Stromverbrauch lässt sich ableiten, dass jeweils alle 3 physischen Sensoren genutzt werden, da die Summe der Einzelstromverbräuche dem Verbrauch jedes einzelnen virtuellen Sensors entspricht.

Beispielsweise normalisiert der lineare Beschleunigungssensor (Linear Acceleration Sensor) die Erdanziehungskraft, welche Teil der gemessenen Beschleunigung ist und auch im ruhenden Zustand immer auf das Gerät einwirkt. So wirkt permanent eine Beschleunigung von 9,81 m/s² auf das Gerät nach unten. Möchte man jedoch nur die Beschleunigung, die abgesehen davon wirkt, für eine Anwendung nutzen, so könnte das Normalisieren mittels Android Programm Code unter Umständen aufwendig werden und zudem Performance kosten. Eine Hardware nahe Implementierung stellt nicht nur eine nützliche, sondern auch eine performante Lösung zu Verfügung.

Wie Sensor Fusion implementiert ist, beschreibt Milette /8/ ab Seite 115. Daraus lässt sich ableiten, dass die Sensor Fusion tendenziell näher dem Android Framework zu zuordnen ist. Daher werden die anderen virtuellen Sensoren und ihre Nutzung im betreffenden Kapitel beschrieben.

Sensor	**Hersteller**	**Bereich**	**Auflösung**	**Stromverbrauch (mA)**
KR3DM 3 axxis Accelero-meter	ST Microelectronics	19,6133(m/s²)	0,019153614(m/s²)	0,23
AK8973 3 axxis Magnetic	Asahi Kasei Micro-	2000,0(µT)	0,0625(µT)	6,8

field sensor	devices			
K3G Gyroscope sensor	ST Microelectronics	34,906586(rad/s)	0,0012217305(rad/s)	6,1
Rotation Vector Sensor	Google Inc.	1,0(unitless)	5,9604645E-8(unitless)	13,13
Gravity Sensor	Google Inc.	19,6133(m/s²)	0,019153614(m/s²)	13,13
Linear Acceleration Sensor	Google Inc.	19,6133(m/s²)	0,019153614(m/s²)	13,13
Orientation Sensor	Google Inc.	360,0(°)	0,00390625(°)	13,13

Tabelle 2: Sensoren Google Nexus S(Sleek Apps)

Die 3 physischen Sensoren – Accelerometer, Magnetic Field Sensor, Gyroscope Sensor - werden im folgenden Kapitel hinsichtlich ihrer grundsätzlichen Funktionsweise erläutert und der Bezug zu den in Tabelle 2: Sensoren Google Nexus S aufgelisteten Sensoren hergestellt.

2.2 Beschreibung von physikalischen Sensoren

In diesem Abschnitt wird die prinzipielle Funktionsweise von Sensoren beschrieben. Dabei werden die physikalischen Grundlagen sowie die Realisierungsformen erklärt. Augenmerk wird auf jene Bauformen gelegt, die in Android Smartphones Verwendung finden.

2.2.1 Accelerometer - Beschleunigungssensoren

Beschleunigungssensoren ermöglichen die Messung der Beschleunigung, die auf einen Gegenstand wirkt. Die Funktionsweise beruht darauf, dass Kraft auf eine Masse wirkt, wenn diese bewegt wird. Diese Kraft wird als Druck messbar.

Die Beschleunigung kann über die Beziehung

$F = m \cdot a$

F … Kraft

m … Masse

a … Beschleunigung

(2. Newton'sche Gesetz) errechnet werden. Wobei F die gemessene Kraft bzw. den Druck darstellt und m die bekannte Probemasse innerhalb des Bauteils, auf den die Kraft wirkt und die Beschleunigung. Siehe dazu /4/ Seite 240.

Um eine Messung der Beschleunigung im Raum zu ermöglichen, muss eine Messung der Krafteinwirkung in 3 orthogonal aufeinander stehende Achsen erfolgen. Dies erfordert dementsprechend 3 Beschleunigungssensoren (vgl. Lechner /6/, Seite 7).

Zusätzlich zur linearen Beschleunigung wirkt permanent die Erdbeschleunigung durch die Anziehungskraft der Erde. Möchte man auf Basis von Beschleunigungsdaten eine Bewegung im Raum messen, muss dieser Effekt normalisiert werden.

Die Umsetzung zur Messung von Beschleunigung kann nach unterschiedlichen Techniken erfolgen. In dieser Ausarbeitung wird ein Ausschnitt daraus erläutert. Augenmerk wird hierbei auf kapazitive Methoden gelegt, da diese bei Smartphones zum Einsatz kommen. Zwei alternative Methoden werden dargestellt, um dem Leser einen Hinweis auf andere Techniken zu geben.

- Kapazitive Beschleunigungssensoren
- Schwingungsbeschleunigungssensoren
- Piezoelektrische Beschleunigungssensoren

Alle 3 werden auf kleinstem Raum in Form von MEMS (Mikro-Elektro-Mechanische Systeme) gebaut. Diese Bauform ist Stand der Technik und als Implementierungsverfahren vorherrschend (/11/, Seite 270).

Bei dem untersuchten Gerät kommt ein kapazitiver MEMS Sensor zum Einsatz. Laut Angabe des Herstellers ST Microelectronics weist dieser eine Größe von 4 x 4 x 1,45 mm (LFCS Package) auf. Dennoch werden im Folgenden alle 3 angeführten Techniken beschrieben, um dem Leser einen Überblick und unterschiedliche Charakteristika zu verdeutlichen.

2.2.1.1 Kapazitive Beschleunigungssensoren

Kapazitive Beschleunigungssensoren arbeiten gem. Hoffmann /4/, Seite 241 nach dem Feder-Masse Prinzip und sind kleine Silizium-Bauteile. Die Auslenkung einer kleinen gefederten Masse auf einem Silizium-Steg, führt zu einer Veränderung der kapazitiven Eigenschaften des Bauteils. Die Veränderung der Kapazität, wenn auch nur sehr gering (Pikofarad), kann für die Ermittlung der Beschleunigung herangezogen werden.

In Abbildung 2: Illustration kapazitiver Beschleunigungssensor wird der Aufbau eines kapazitiven Beschleunigungsmesser stark vereinfacht dargestellt. Die in der Mitte angebrachte Probemasse schwingt entlang der sensitiven Achse und beeinflusst durch die Auslenkung die gemessene Spannung aufgrund der sich verändernden Kapazität.

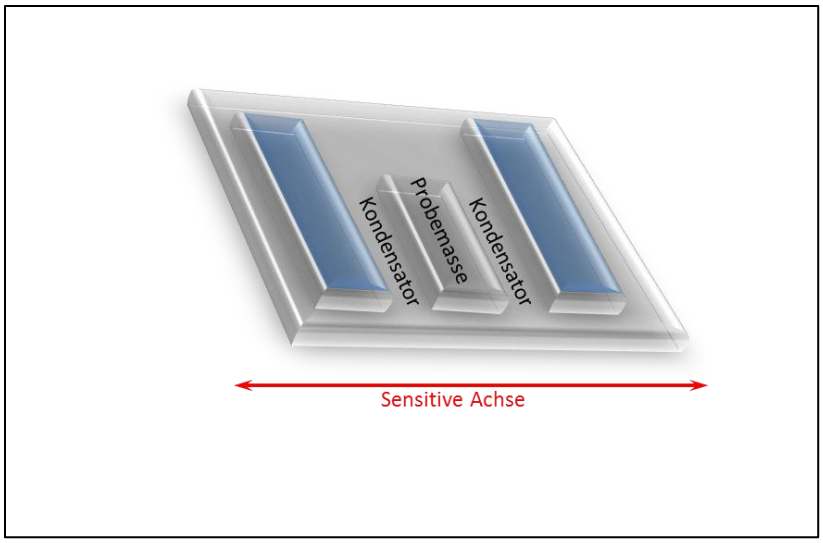

Abbildung 2: Illustration kapazitiver Beschleunigungssensor(Quelle: Hoffmann)

Die einzelnen Schritte ausgehend von einer Beschleunigung hin zu einem elektrischen Signal sind in Abbildung 3: Kapazitives Messen von Beschleunigung (Quelle: Reif) dargestellt.

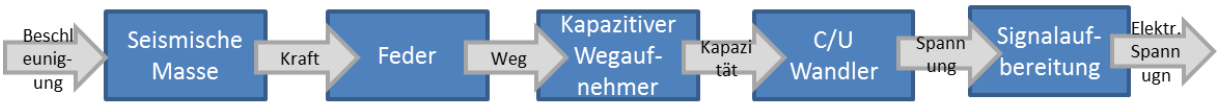

Abbildung 3: Kapazitives Messen von Beschleunigung (Quelle: Reif)

2.2.1.2 Schwingungsbeschleunigungssensoren

Schwingungsbeschleunigungssensoren arbeiten mit Quarzen, die unter Druck ihre Schwingungsfrequenz verändern. Die Schwebungsfrequenz – die Überlagerung von 2 Frequenzen – von 2 Quarzen, die entlang der Bewegungsrichtung vor und nach der Masse angebracht sind, gibt Auskunft über die Beschleunigung.

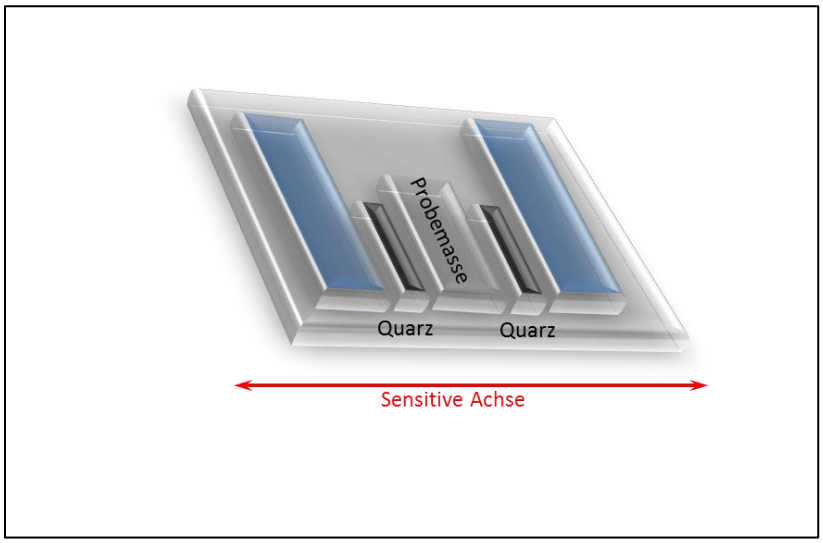

Abbildung 4: Illustration Schwingungsbeschleunigungssensor (Quelle: Wendel)

Abbildung 4: Illustration Schwingungsbeschleunigungssensor zeigt den prinzipiellen Aufbau eines Schwingungsbeschleunigungssensors. Die Quarze sind fest zwischen Probemasse und Rahmen montiert. Die Bewegung entlang der sensitiven Achse übt Druck oder Zug auf die Quarze aus, was zu unterschiedlichem Schwingungsverhalten führt und Rückschluss auf die Beschleunigung ermöglicht. Wendel beschreibt diesen in /12/ Seite 68 als Vibrating Beam Beschleunigungsmesser.

Abbildung 5: Messen von Beschleunigung mittels Schwingung (Quelle: Reif)

2.2.1.3 Piezoelektrische Beschleunigungssensoren

Bei Piezoelektrischen Beschleunigungssensoren wird die Veränderung des Drucks auf eine Masse in Form eines erzeugten Stroms gemessen (vgl. Reif /10/, Seite 122).

Die Probemasse ist bekannt und die Krafteinwirkung wird als Spannung aus den piezoelektrischen Elementen abgeleitet. Abbildung 6: Illustration Piezoelektrischer Beschleunigungssensor vermittelt einen Eindruck zur Funktionsweise. Die Piezoelemente sind zwischen der Probemasse und dem Rahmen schlüssig angebracht. Ein Druck auf ein Piezoelement erzeugt Strom in diesem, der für die Messung verwertet wird. Genau genommen wird lediglich die Änderung der Krafteinwirkung in Strom umgewandelt, was gleichzeitig das Einsatzgebiet einschränkt.

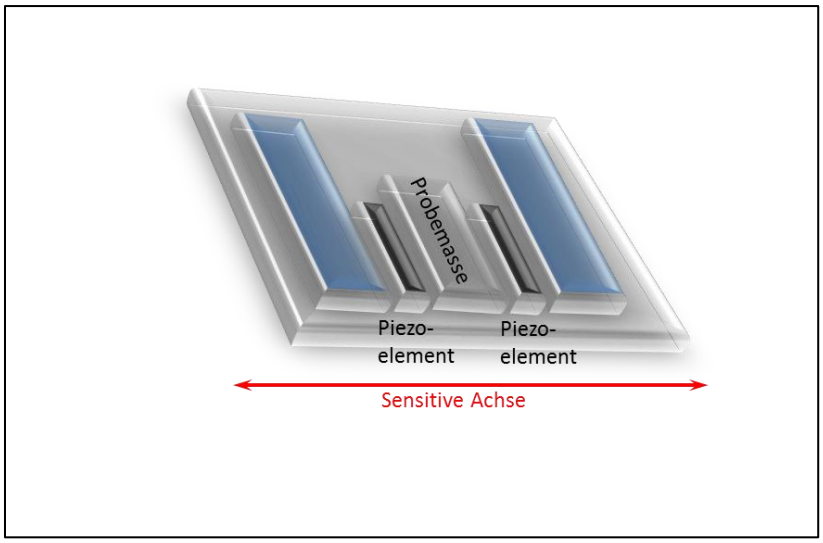

Abbildung 6: Illustration Piezoelektrischer Beschleunigungssensor (Quelle: Reif)

Beschleunigungssensoren nach diesem Prinzip können nur die Veränderungen der Beschleunigung, etwa durch auftretende Unwucht in einem Rad, wahrnehmen und eignen sich daher für die Messung der Beschleunigung durch Bewegung nicht. Das Funktionsprinzip ist jedoch einfach, was durch Abbildung 7: Piezoelektrisches Messen von Beschleunigung (Quelle gezeigt wird. Die geringe Anzahl an Bauelementen und dadurch geringere Komplexität verglichen mit dem kapazitiven Messen ist hierbei sichtbar.

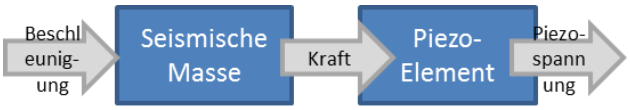

Abbildung 7: Piezoelektrisches Messen von Beschleunigung (Quelle: Reif)

2.2.2 Gyroscope Sensor - Drehratensensor

Drehratensensoren, auch Gyroskope oder Kreisel genannt, ermöglichen die Messung von Rotationsbewegungen entlang einer Achse. Um eine Messung von Drehbewegung eines Gegenstandes im Raum zu ermöglichen, sind wieder 3 orthogonal aufeinander stehende Achsen zu berücksichtigen. Die Bewegungen entlang der unterschiedlichen Achsen werden wie folgt bezeichnet.

- Gierrate bedeutet die Drehung um die Hochachse
- Nickrate bedeutet die Drehung um die Querachse
- Rollrate bedeutet die Drehung um die Längsachse

Zur Messung wird im Wesentlichen auf 2 physikalische Prinzipien zurückgegriffen. Einerseits die Coriolis-Kraft und andererseits der Sagnac-Effek. Nachdem für die gegenständlichen Drehratensensoren MEMS Sensoren eingesetzt werden, die die Coriolis-Kraft nutzen, wird darauf Augenmerk gelegt. Der Sagnac-Effekt wird als alternative Möglichkeit dargestellt.

2.2.2.1 Coriolis-Kraft

Die Coriolis-Kraft ist wie die Zentrifugalkraft eine Trägheitskraft, die auftritt, wenn sich ein Körper in einem rotierenden System bewegt, beschreibt Eichler in /2/ auf Seite 27. Wird etwa eine Masse in einem drehenden System geworfen, so beschreibt die Flugbahn bezogen auf das ruhende System eine Gerade, bezogen auf das rotierende System eine gekrümmte Kurve. Die Coriolis-Kraft, die ausschließlich in rotierenden Bezugssystemen auftritt, beschreibt, welche (Schein-)Kraft auf einen Körper wirkt und wie die Bahnbewegung manipuliert wird, so Schiessle /11/, Seite 283. Dies ist ein Grund dafür, warum die Coriolis-Kraft auch in der Meteorologie genutzt wird. Es können damit Wolkenbewegungen und Wetterentwicklungen vorhergesagt werden.

Die Coriolis-Kraft lässt sich als Formel wie folgt darstellen:

$F_c = 2 . m . v . \omega$

F_c …Coriolis Kraft

m …Probemasse

v …Geschwindigkeit

ω … Winkelgeschwindigkeit

.

Die Detektion der Kraft erfolgt, laut Reif /10/, Seite 127, meist nach kapazitiven Verfahren, etwa im MEMS-Kreisel (vgl. Hoffmann /4/, Seite 62). So wie in Abbildung 8: Illustration kapazitives Gyroskop (Coriolis-Kraft) verdeutlicht, erfolgt eine Anregung, etwa durch einen piezoelektrischen Aktor, einer Probemasse zu einer Primär-Schwingung mit der Geschwindigkeit v. Unterliegt der Körper einer Drehung mit der Winkelgeschwindigkeit ω verändert sich die Auslenkung der Probemasse entlang der sekundären Schwingungsrichtung. Die Kapazitätsänderung wird analog zum Beschleunigungssensor gemessen. Aus der Kapazitätsänderung kann auf die Winkelgeschwindigkeit ω rückgeschlossen werden. Beschrieben in Schiessle /11/, auf Seite 284. Demnach wird anders als bei Beschleunigungssensoren bei Drehratensensoren die Geschwindigkeit gemessen und nicht die Beschleunigung.

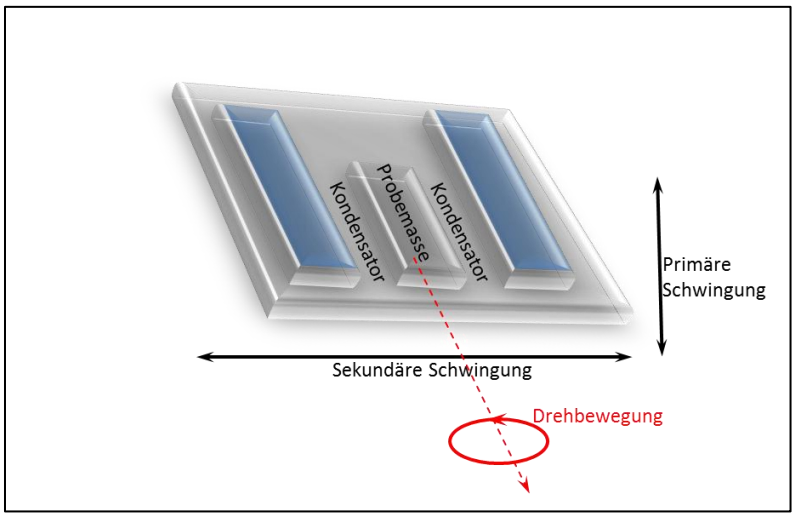

Abbildung 8: Illustration kapazitives Gyroskop (Coriolis-Kraft) (Quelle: Schiessle)

In der Praxis gibt es hierzu unterschiedlichste Bauformen. Die Anregung zur Primärschwingung kann etwa piezoelektrisch, elektrostatisch oder elektromagnetisch erfolgen, die Sensorik der Krafteinwirkung kann piezoresistiv oder kapazitiv erfolgen.

Die Primärschwingung, die permanent durch Energiezufuhr angeregt werden muss, führt auch zu dem in Tabelle 2: Sensoren Google Nexus S ersichtlichen, deutlich höheren Stromverbrauch im Vergleich zum Beschleunigungssensor.

2.2.2.2 Sagnac-Effekt

Der Sagnac-Effekt wird genutzt um Rotationsbewegungen, laut Henn /3/, Seite 9, mittels optischer Erscheinungen zu messen. Dabei laufen Lichtstrahlen in einem Ringinterferometer in entgegengesetzter Richtung. Am Auskoppelpunkt findet eine Überlagerung der gegenlaufenden Lichtstrahlen statt. Wird der Ring gedreht führt dies zu unterschiedlichen Lichtlaufzeiten und die Drehbewegung wird als Verschiebung der Interferenzstreifen sichtbar. Dieser Effekt wird beispielsweise beim, von Wendel /12/ auf Seite 63 beschriebenen, Faserkreisel genutzt. Mit dem relativistischen Geschwindigkeitsadditionsgesetz kann ein Rückschluss auf den Drehwinkel hergestellt werden. Die Detektion erfolgt mittels Fotodiode.

Drehratensensoren auf Basis des Sagnac-Effekts ermöglichen genaueste Resultate, hier im Besonderen der Ringlaser-Kreisel. Hierbei wird Laserlicht genutzt und durch Spiegel in einem Dreieck in jeweils gegengesetzte Richtungen gelenkt (vgl. Wendel /14/, Seite 66).

2.2.3 Messfehler bei Bewegungssensoren

Sowohl die Messung von Beschleunigungen als auch jene von Drehraten unterliegen Messfehlern, welche in diesem Kapitel näher betrachtet werden.

Die Ursache von Messfehlern begründet sich durch kleine Abweichungen im Zuge der Produktion und im Betrieb durch äußere Faktoren, wie etwa Temperaturschwankungen. Eine wesentliche Ursache begründet Wendel /14/, auf Seite 24, in der Nichtlinearität von realen Systemen.

Messfehler sind durch folgende 3 Faktoren beeinflusst, Abbildung 9: Bias und Skalenfaktorfehler (Quelle: Wendel) stellt dies grafisch dar:

- Skalenfaktorfehler
- Bias
- Sensorinherentes Rauschen

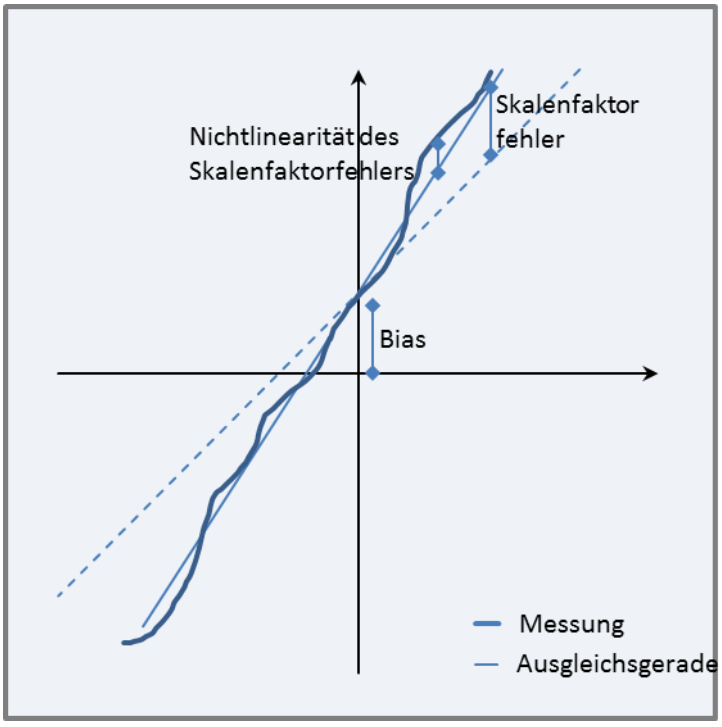

Abbildung 9: Bias und Skalenfaktorfehler (Quelle: Wendel)

Produktionsbedingte Abweichungen werden durch Misalignment Matrizen für die Sensor-Triade – für die Messung im Raum sind 3 orthogonale Achsen zu betrachten und daher 3 Sensoren erforderlich – beschrieben. Damit wird der Skalenfaktorfehler, bestehend aus einem konstanten und einem bewegungsabhängigen Fehler, ausgedrückt, welcher nach Vermessung des Sensors durch die Elektronik rechnerisch kompensiert wird (vgl. Wendel /14/, Seite 69). Der Skalenfaktorfehler unterliegt zudem dem Problem der Nichtlinearität,

welche ebenso berücksichtigt werden muss. Dies wird an dieser Stelle erwähnt, aber nicht näher spezifiziert.

Bias drückt die Abweichung vom Nullpunkt aus. Dies führt dazu, dass Bewegung oder Drehung auch dann detektiert werden, wenn sich das Gerät real in einer Ruheposition befindet. Abhängig von der Qualität werden Abweichungen in Grad, bei Gyroskopen, oder Geschwindigkeit, bei Beschleunigungssensoren, pro Stunde oder Sekunde ausgedrückt. Üblicherweise bleibt der Bias nach dem Einschalten konstant, was im Betrieb durch die Rechenlogik kompensiert werden kann, kann aber auch durch äußere Einflüsse variieren. Über den zeitlichen Verlauf ergibt sich daraus ein Bias-Drift.

Sensorinherentes Rauschen wird als weiß, normalverteilt und mittelwertfrei angenommen (Wendel /14/, Seite 70). Und resultiert in weiterer Folge bei Kreisel in einem Winkelfehler mit einer bestimmten Standardabweichung und wird als Angle Random Walk ausgedrückt. Bei Beschleunigungssensoren entsteht eine Abweichung hinsichtlich Geschwindigkeit, welche als Velocity Random Walk bezeichnet wird.

Die Tabelle 3: Messfehler bei Gyroskopen und Beschleunigungssensoren (Quelle: Wendel) gibt eine Übersicht über die auftretenden Messfehler bei Bewegungssensoren und die Einheiten zur Charakterisierung wieder. Angeführt sind hierbei die verwendeten MEMS Bauteile, aber zu Vergleichszwecken auch genauere Implementierungsformen.

Im Bereich der Gyroskope ist der Ringlaserkreisel deutlich genauer. Ebenso gilt dies im Vergleich zwischen Vibrating Beam und Pendel. Das MEMS Pendel steht hier für das Feder-Masse-Prinzip mit kapazitiver Messung, welches in Android Smartphones wahrscheinlich zur Anwendung kommt. Wahrscheinlich deshalb, weil keine eineindeutigen Informationen diesbezüglich gefunden werden konnten, aber aus den vorhandenen Informationen der Schluss gezogen werden kann. Es wird daher davon ausgegangen, dass die unter MEMS Gyroskop und MEMS Pendel angeführten Qualitätsattribute für weitere Überlegungen im Zusammenhang mit Android Smartphones als Grundlage dienen können. Es zeigt aber auch, dass unabhängig von produktions- oder kostenbedingten Notwendigkeiten, weitaus genauere Messmöglichkeiten grundsätzlich zur Verfügung stehen.

Fehler	**MEMS Gyroskop**	**Ringlaserkreisel**	**MEMS Pendel**	**MEMS Vibrating Beam**
Bias	5°/h bis 5°/s	0,001 bis 10 °/h	0,1 bis 10 mg	0,1 bis 1 mg
Skalenfaktorfehler	>400 ppm (parts per million)	5 ppm	1000 ppm	100 mp
Angle/Velocity Random Walk	1°/√h	0,01°/√h	0,04 $^{m/s}$/√h	0,01 $^{m/s}$/√h

Tabelle 3: Messfehler bei Gyroskopen und Beschleunigungssensoren (Quelle: Wendel)

2.2.4 Magnetic Field Sensor – Magnetischer Feldsensor

Mit dem Magnetischen Feldsensor wird das, durch den Sensor strömende, magnetische Feld erfasst. Daraus ergeben sich viele Anwendungsgebiete mit unterschiedlichsten Implementierungsmethoden. Ein großer Anwendungsbereich ist die Materialprüfung, beschreibt Schiessle /11/, Seite 165.

Der magnetische Feldsensor ist auch ein wesentlicher Baustein zur Implementierung eines elektronischen Kompasses. Magnetische Feldsensoren in Smartphones arbeiten nach dem Hall-Effekt. Bevor der Hall-Effekt und die gegenständliche Sensorik betrachtet werden, erfolgt eine kurze Erläuterung zum magnetischen Feld der Erde. Es verläuft durch die Längsachse der Erde entsprechend der Abbildung 10: Geomagnetisches Feld (Quelle: Internet), tritt jeweils ungefähr bei den Polkappen aus und verläuft durch das Weltall und an der Erdoberfläche entlang zur anderen Polkappe. Magnetischer Pol und geografischer Pol stimmen nicht gänzlich überein. Diese Abweichung wird, ausgenommen in Pol-nahen, Gebieten vernachlässigt. Interessant ist dabei auch, dass der magnetische Südpol im Norden liegt und umgekehrt.

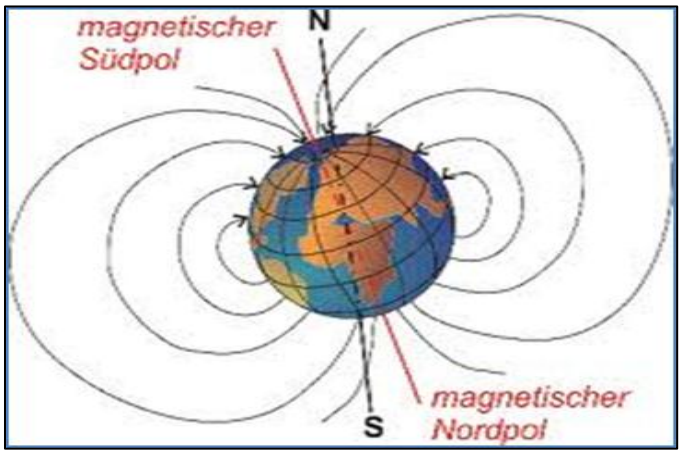

Abbildung 10: Geomagnetisches Feld (Quelle: Internet)

Der Hall-Effekt beruht auf der Lorentzkraft, die auftritt wenn sich Ladungsträger durch einen Leiter bewegen und senkrecht zur Bewegungsrichtung ein Magnetfeld angelegt wird. Auf die Ladungsträger wirkt dann eine Kraft orthogonal zum Magnetfeld und zur Bewegungsrichtung. Es findet eine Ladungsverschiebung statt, die ein elektrisches Feld erzeugt. Die so entstehende Hall-Spannung ist direktproportional zur Lorentzkraft und kann gemessen werden (vgl. Schiessle /11/, Seite 178). Abbildung 11: Prinzipaufbau Hall-Element (Quelle: Schiessle) zeigt den Aufbau eines Hall-Elementes. Dabei wird das Hall-Element entlang der Y-Achse mit einem konstanten Steuerstrom durchflossen. Entlang der Z-Achse durchläuft das magnetische Feld das Hall-Element. Als Reaktion auf den magnetischen Fluss verschieben sich die Ladungsträger und die Messung der Hall-Spannung entlang der X-Achse wird ermöglicht.

Die wirkende Lorentz-Kraft kann vereinfacht durch folgende Formel dargestellt werden.

$F_L = q \cdot v_y \cdot B_z$

F_L … Lorentz Kraft

q ... Elektrische Ladung

v_y…Geschwindigkeit der Ladungsträger

B_z…Magnetische Flussdichte in Z-Richtung

Für Smartphones werden AMR-Sensoren (Anisotropic-Magnetic-Resistenc-Effekt) eingesetzt, beschrieben von Schiessle /11/, Seite 205. Hierbei wird der Effekt genutzt, dass die Richtung des Magnetfeldes den elektrischen Widerstand beeinflusst. Im gegenständlichen Sensor werden 3 Achsen zur Ermittlung des magnetischen Erdfeldes genutzt.

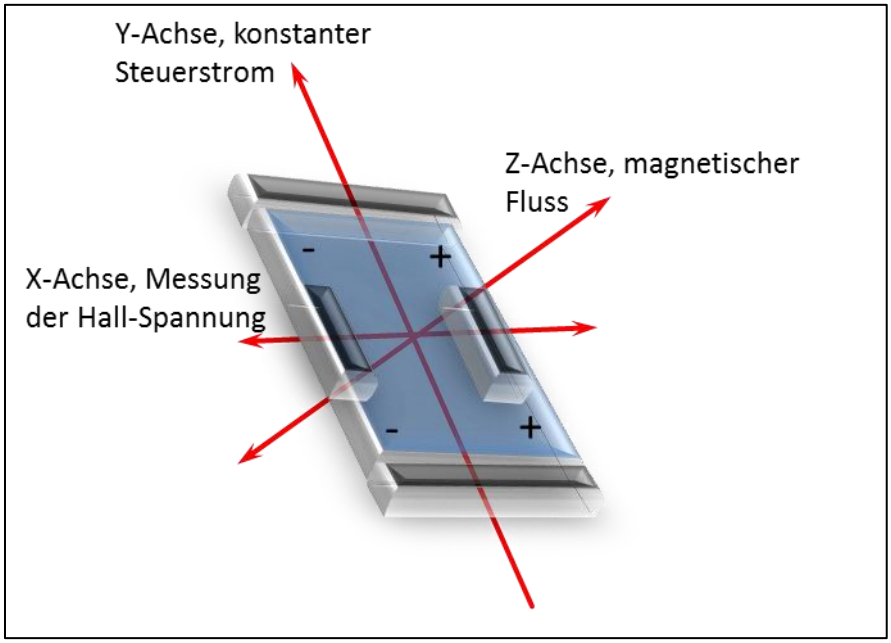

Abbildung 11: Prinzipaufbau Hall-Element (Quelle: Schiessle)

Damit lassen sich darüber hinaus auch Lage- und Orientierungsaufgaben lösen. In jedem Fall ergibt sich im Rahmen der Sensor Fusion eine vorteilhafte Ergänzung.

2.3 Navigation mit Sensoren

Auf Basis der Sensoren, wie sie in den vorangegangenen Kapiteln dargestellt wurden, lassen sich Navigationsaufgaben lösen. Dieses Kapitel beschäftigt sich mit dem Thema Trägheitsnavigation, beschreibt die Begriffe und die theoretischen Grundlagen.

2.3.1 Trägheitsnavigation und inertiale Navigation

Wichtige Begriffe im Zusammenhang mit Navigation mittels Bewegungs- und Drehratensensoren sind Trägheitsnavigation oder inertiale Navigation, welche synonym verwendet werden. Im Prinzip handelt es sich dabei um eine Koppelnavigation, bei der die Bewegungsrichtung und die Geschwindigkeit permanent gemessen werden. Durch Betrachtung des Zeitintervalls kann die Positionsänderung seit dem letzten bekannten Ort ermittelt werden (vgl. Wendel /14/, Seite 27). Das Koppelverfahren wurde schon vor mehr als 200 in der Schifffahrt genutzt, allerdings mittels Log und Kompass.

Bei Trägheitsnavigationssystemen werden die Beschleunigungen und Drehraten genutzt, um die nötigen Informationen zu erhalten. Entlang der 3 Achsen können 3 Rotationsbewegungen auftreten. Dies erfordert daher die Berücksichtigung von 6 Bewegungsmöglichkeiten. Abbildung 12: Bewegungs- und Drehachsen veranschaulicht die 6 Bewegungsmöglichkeiten. In weiterer Folge werden demnach 3 Beschleunigungssensoren und 3 Drehratensensoren benötigt, die orthogonal aufeinander gerichtet sind.

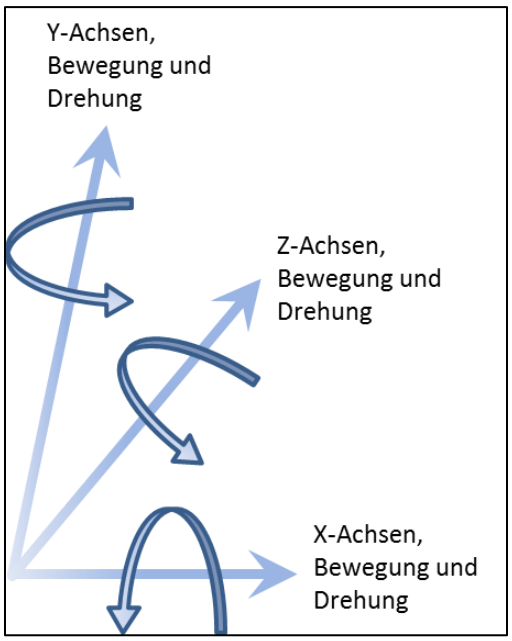

Abbildung 12: Bewegungs- und Drehachsen

2.3.2 Strapdown-Systeme

Seit den 60er Jahren gibt es Strapdown-Systeme (Wendel /14/ beschreibt diesen auf Seite 27 und 45), die dadurch charakterisiert sind, dass die Inertialsensor-Einheit fest mit einem Fahrzeug verbunden ist. Beim Strapdown-Algorithmus handelt es sich um die Rechenvorschriften, wie aus den gemessenen Beschleunigungen und Drehraten die neue Position, ausgehend von einer letzten, bekannten Position errechnet wird. Zur Berechnung der Lage muss die Drehrate einmal integriert werden und für die Position die Beschleunigung zweimal integriert werden.

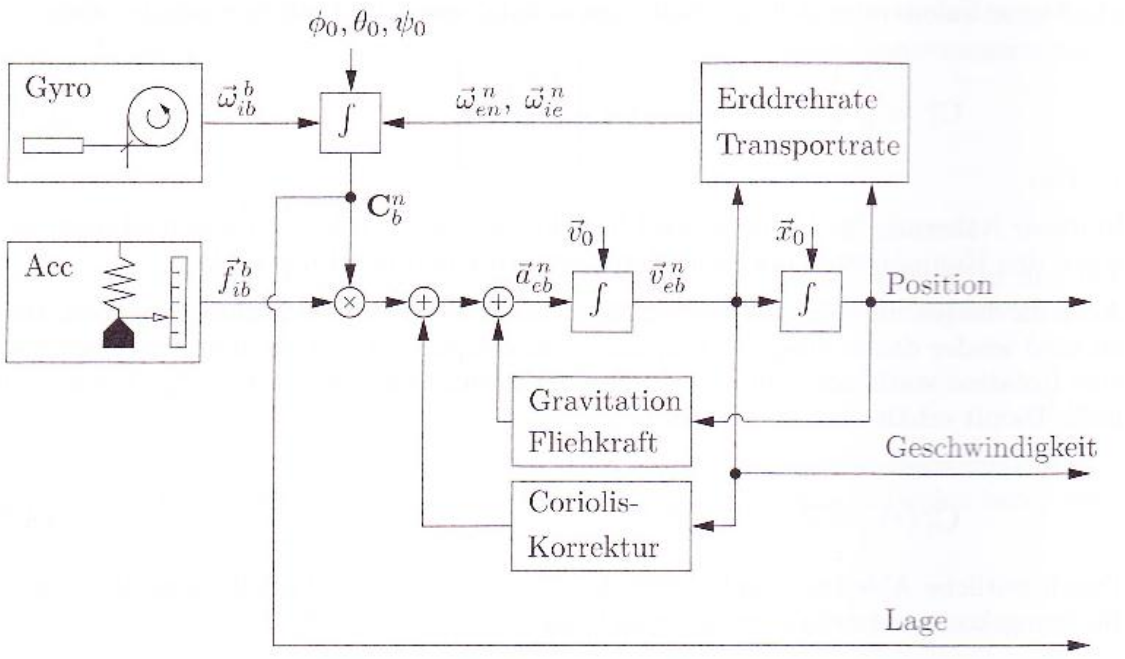

Abbildung 13: Strapdown Algorithmus (Quelle: Wendel]

Ausgehend von den erwähnten Prinzipien ist die Möglichkeit zur Trägheitsnavigation, mittels der in dem gegenständlichen Smartphone eingebauten Sensoren, grundsätzlich gegeben.

Ein wesentliches Kriterium von Strapdown-Systemen kann aber im Normalfall beim Einsatz von Smartphones nicht gewährleistet werden. Und zwar, dass dieses fix mit einem Fahrzeug oder mit dem Objekt, das gemessen werden soll, verbunden ist. Dies bedeutet erheblichen Mehraufwand bei der Berechnung, möchte man die Navigation durch ein Smartphone, das etwa in der Manteltasche liegt, sicherstellen. Jeder Schritt müsste kompensiert werden und jede Lage des Smartphones in eine neutrale Lage umgewandelt werden.

2.3.3 Erwartete Praxistauglichkeit und Optimierungsmöglichkeiten

Wie bereits im Kapitel 2.2.3 Messfehler bei Bewegungssensoren dargestellt, kommt es zu einer Reihe von Messfehlern, die dazu führen, dass die zuverlässige Nutzung eingeschränkt wird. Zusätzlich zu den Messfehlern quadrieren sich die Auswirkungen durch die Bildung von doppelten Integralen.

Eine Kombination von mehreren Quellen zur Ortung und Festlegung der Position ist in jedem Fall erforderlich, um brauchbare Genauigkeit über einen längeren Zeitraum zu erzielen. Dies kann einerseits durch die Sensor Fusion erzielt werden oder etwa durch die Nutzung von Wireless LAN Signalen, deren Laufzeiten zum Accesspoint gemessen werden können, und im besten Fall GPS. Auch Luftdrucksensoren können als Ergänzung genutzt werden, um Höhenbewegungen zu ermitteln. Die so erzeugten Integrierten Navigationssysteme steigern die Zuverlässigkeit der Ergebnisse deutlich.

Verbesserungen der Ergebnisse können auch durch Stochastische Verfahren erzielt werden. Kalman-Filter in unterschiedlichen Ausprägungen können hierfür genutzt werden. Dies wird

von Wendel /14/ auf Seite 117 und 129 beschrieben. Da es sich hierbei wieder um ein weites Wissensgebiet handelt, wird dies nur erwähnt, aber nicht weiter vertieft.

3 Android und die Nutzung von Sensoren

Das Ziel dieses Kapitels ist es, dem Leser das Rüstzeug für eine Nutzung von Android zu geben. Dazu wird im ersten Schritt Android und der Aufbau allgemein beschrieben, sowie eine Anleitung zur Installation der Entwicklungsumgebung geboten. Anschließend erfolgt eine Beschreibung der Verwendung der Sensoren. Dies umfasst auch die nötige Syntax.

Die bereits im vorangegangenen Kapitel beleuchteten Sensoren werden in diesem Kapitel ausschließlich aus der Sicht von Android erklärt.

3.1 Einleitung zu Android

Android ist sowohl ein Betriebssystem für Smartphones als auch eine Software-Plattform (vgl. Android Developer Site /1/). Im Jahr 2008 brachte die Open Handset Alliance, deren Hauptmitglied Google ist, die erste Version auf den Markt. In nur wenigen Jahren hat Android große Verbreitung gefunden und dominiert derzeit bei den eingesetzten Betriebssystemen für Smartphones den Markt.

Dies liegt zum einen daran, dass es sich um eine freie Software handelt, deren Quellcode offengelegt wurde. Und zum anderen an der einfach zugänglichen Entwicklungsumgebung. Die Entwicklung erfolgt mittels Eclipse und Android Software Development Kit (SDK), die Programmierung erfolgt in Java. Zusammen mit der weitverbreiteten Programmiersprache und den leicht zugänglichen, unterstützenden Informationen zur Softwareentwicklung, ergibt dies eine sehr beliebte Plattform für Softwareentwickler.

3.2 Konzeptioneller Aufbau von Android

3.2.1 Android Framework

Die Developer Site von Android /1/ bietet umfangreiche Hilfestellung bei der Entwicklung von Android Anwendungen und erläutert die ersten Schritte. Die in diesem Kapitel gesammelten Informationen stammen Großteils von Android direkt. Einen guten Überblick über das Framework bietet die Abbildung 14: Aufbau von Android (Quelle: Android).

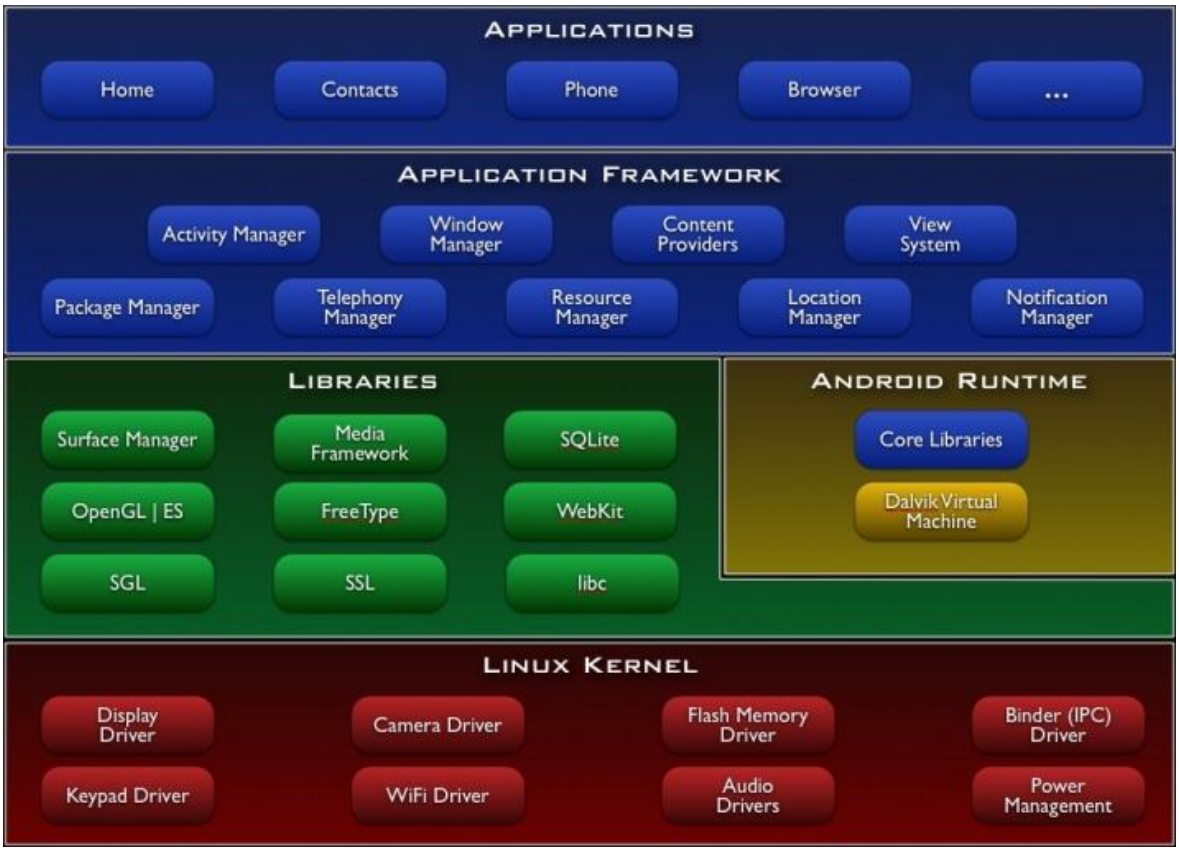

Abbildung 14: Aufbau von Android (Quelle: Android)

Android wird mit einer Reihe von Kernapplikationen, wie Adressbuch, Telefon, Browser, Email-Client, SMS Programm, Kalender, ausgeliefert. Alle Applikationen sind in Java entwickelt.

Mit dem Applikations Framework wird eine offene Entwicklungsumgebung bereitgestellt. Diese ermöglicht es Entwicklern innovative und anspruchsvolle Applikationen zu entwickeln, die alle Möglichkeiten, die auch den Kernapplikationen zur Verfügung stehen, nützen.

Android beinhaltet ein Set von C/C++ Libraries, die von allen Komponenten des Systems genutzt werden können. Jede Android Applikation läuft in einer abgesicherten Umgebung, in der Dalvik Virtual Machine. Das Betriebssystem von Android ist ein Multi-User Linux System, bei der jede Applikation ein eigener User ist.

Wie schon erwähnt, sind Android Applikationen in Java programmiert. Das Android SDK Werkzeug exportiert den Code in ein Android Package mit der Extension *.apk. Diese Datei kann auf das Android Smartphone gebracht werden, etwa als Mail Anhang, und wird von Android als installierbare Applikation erkannt und als solche behandelt.

Im Normalfall hat jeder Prozess seine eigene virtuelle Maschine (VM) und ist damit isoliert von anderen Anwendungen. Android stellt das Prinzip des „principle of least privilege“ sicher, das eine sehr sichere Umgebung ermöglicht, bei der die Applikation nur über die Berechti-

gungen verfügt, die es erteilt bekommt. Dies führt auch zu einem Schutz zwischen den Applikationen.

Eine Android Anwendung besteht aus einigen essentiellen Komponenten, auf die hier eingegangen wird.

Activities stellen einen Ein-/Ausgabebildschirm dar, mit dem die Interaktion zwischen Anwender und System erfolgen kann. Innerhalb der Entwicklungsumgebung, dem Android SDK, kann das Layout definiert werden und die jeweiligen Ein-/Ausgabekomponenten eingefügt werden. Ein Vorteil der Android Umgebung, die die Weiterverwendung von bestehenden Applikationen oder deren Komponenten gut unterstützt.

Services sind Prozesse, die Hintergrundverarbeitung oder Aufgaben für Remote-Prozesse übernehmen. Damit kann Verarbeitung, die keine direkte Anwenderinteraktion benötigt, realisiert werden.

Content Providers ermöglichen die Sammlung und Bereitstellung von Anwendungsdaten. Die Daten können in einer Datenbank – SQLite Datenbank -, im Web oder in jedem anderen zuverlässigen Speicher abgelegt sein. Durch den Content Provider können auch andere Applikationen auf die Daten zugreifen und auch verändern, sofern die Berechtigungen vorliegen.

Broadcast Receivers ist eine Komponente die systemweite Informationen und auch Ankündigungen empfängt und bereitstellt. Die Aufgabe dieser Komponente lässt sich am Beispiel eines niedrigen Akkustandes erklären. Wenn die Meldung eines zu niedrigen Akkustandes gebroadcastet wird, wird es von dem Broadcast Receiver empfangen und angezeigt.

Mittels **Intents** werden Activities, Services und Broadcast Receivers aktiviert. So können während der Laufzeit individuelle Komponenten zusammengefügt werden. Unabhängig davon, ob die Komponenten Teil der eigenen Applikation sind oder einer anderen. Die Aufgabe von Intents kann so verstanden werden, dass man die Absicht hat eine bestimmte Funktion zu nutzen. Aufgrund des Berechtigungssystems kann eine Komponente einer anderen Applikation nicht direkt angesprochen werden, das Betriebssystem übernimmt diese Aufgabe.

Die **Manifest**-Datei ist eine xml Datei, welche zentrale Informationen zur Applikation beinhaltet. Sie muss im Rootverzeichnis liegen und heißt immer AndroidManifest.xml. Im Manifest werden alle Komponenten beschrieben. Darüber hinaus werden Berechtigungen geregelt und Einschränkungen hinsichtlich Hardware- und Softwarekompatibilität vermerkt.

3.2.2 Ablauf von Android Anwendungen

Android Anwendungen können unterschiedliche Stati einnehmen. Android ist Plattform für unterschiedlichste Anwendungen. Neben der Nutzung für individuelle Anwendungen, steht noch immer jene zur Kommunikation im Vordergrund. Etwa bei einem eingehenden Anrufe wird klar, dass die hier erforderliche Prioritätensetzung und die Handhabung der gerade laufenden Anwendung einige Vorkehrungen erfordern.

Folgende Stati können auftreten, welche auch innerhalb einer Android Anwendung explizit zur Ansteuerung von Routinen genutzt werden. Konkret werden die Activities in unterschiedliche Zustände versetzt.

Active ist jene Anwendung, die im Vordergrund ist und gerade Benutzereingaben entgegennimmt oder Daten bereitstellt. Android wird danach trachten diese Anwendung immer am Leben zu erhalten und gegebenenfalls andere Anwendungen tiefer im Stack zu stoppen, um erforderliche Systemressourcen bereitstellen zu können.

Sobald eine andere Applikation aktiv wird, wird die gerade laufende Anwendung in den **Paused** Mode versetzt. Die Applikation läuft zwar weiter, wird auch noch angezeigt, erhält aber keine Benutzereingaben mehr. Dieser Fall tritt auf, wenn die Applikation in einem Subfenster läuft, das nicht aktiv ist.

Ist die Anwendung nicht mehr sichtbar, wird sie gestoppt. Bei **Stopped** bleibt die Anwendung im Speicher, kann aber bei Ressourcenbedarf vom Betriebssystem jederzeit beendet werden. Dazu ist es wichtig, den Verarbeitungsstatus zu sichern.

Nach Beendigung ist die Anwendung **Inactive**. Die Darstellung Abbildung 15: Ablauf einer Android Anwendung (Quelle: Meier) illustriert den Ablauf.

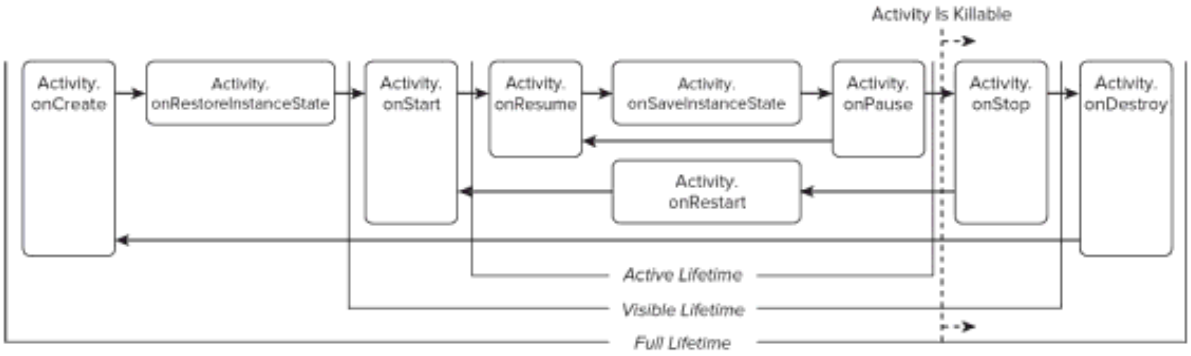

Abbildung 15: Ablauf einer Android Anwendung (Quelle: Meier)

3.2.3 Installation der Android Entwicklungsumgebung

In diesem Kapitel wird beschrieben, wie man eine Android Applikation entwickelt und wie diese auf das Android Smartphone gelangt. Die Entwicklungsumgebung für Android ist unkompliziert, einfach und gratis zu beziehen.

Verfolgt man den Link: http://developer.android.com/sdk/index.html findet man alle erforderlichen Informationen und Programme zur Installation der Entwicklungsumgebung.

Bevor das SDK installiert wird, muss unter Umständen das JDK, Java Development Kit, installiert werden. Möchte man in Eclipse entwickeln, was zu empfehlen ist, so muss dies ebenfalls vorher installiert werden. Damit sind die Vorkehrungen für die nächsten Schritte getroffen und es kann mit dem SDK fortgesetzt werden.

Dazu wird das SDK Starter Package heruntergeladen und installiert. Dieses ist für Windows, MacOS und Linux verfügbar. Für die Entwicklung in Eclipse ist zusätzlich das ADT Plugin erforderlich.

Damit ist die Installation der Entwicklungsumgebung abgeschlossen. Die Installation verläuft zwar in mehreren Teilschritten, hält man sich an die Anleitung wie von Android vorgegeben, so stellte es keine große Herausforderung dar.

Das Testen von Android Applikationen wird durch Emulatoren deutlich erleichtert. Mit dem Android Virtual Device Manager können unterschiedliche Android Smartphones emuliert werden. Die Hersteller bieten entsprechenden Support an. In der angeführten Tabelle 4: Internetlinks zu Emulatorensind die Links zu den Quellen angegeben. Im Normalfall dienen die Emulatoren zum Testen einfacher Funktionen. Einige Änderungen bei Umgebungsparametern, wie etwa ein Anrufeingang, können abgebildet werden. Komplexere, wie die Wirkung von Bewegung des Geräts, in der Regel nicht. Der 3. Link in der Tabelle könnte hierfür eine Hilfestellung bieten. Dies wurde jedoch nicht näher untersucht und kann daher aus der Sicht der Praxistauglichkeit nicht reflektiert werden.

Beschreibung	Link
Samsung Smartphone Emulatoren	http://innovator.samsungmobile.com/cms/cnts/knowledge.detail.view.do?platformId=1&cntsId=9591&linkType=0&nacode=uk&sortType=0&codeType=C514
Samsung Nexus S Skin	http://www.androidacademy.com/3-tutorials/43-hands-on/237-nexus-s-emulator-skin
Samsung Sensor Simulator	http://innovator.samsungmobile.com/cms/cnts/knowledge.detail.view.do?platformId=1&cntsId=9460&listReturnUrl=http%3A%2F%2Finnovator.samsungmobile.com%3A80%2Fplatform.main.do%3FplatformId%3D1&linkType=0&nacode=&codeType=C511

Tabelle 4: Internetlinks zu Emulatoren

3.2.4 Bereitstellen von Applikationen

Sobald die Applikation innerhalb der Entwicklungsumgebung fertiggestellt und ausführlich getestet wurde, kann diese für das sogenannte Publishing vorbereitet werden.

Dazu ist es wichtig, dass das Logging und der eventuell eingeschaltene Debugging Mode deaktiviert werden und alle anderen von Android empfohlenen Bereinigungsmaßnahmen durchgeführt wurden.

Durch die Exportfunktion und der Angabe, dass eine Android Anwendung erstellt werden soll, wird die APK-Datei erstellt. Dieses kann in der einfachsten Form als Mail Anhang an das

Android Smartphone gesendet werden. In der Regel wird allerdings eine Publizierung über einen Google Market Place vorzuziehen sein.

Jede APK-Datei wird mit einem Zertifikat des Autors versehen. Dies kann auch ein privates Zertifikat sein, das im Zuge der Erstellung der APK-Datei generiert wird.

3.3 Sensorik in Android

In diesem Kapitel wird das Ansprechen und Auswerten von Sensordaten behandelt. Der Fokus liegt dabei auf jenen Sensoren, die einen Beitrag zur Aufgabenstellung Navigation leisten können.

Das Kapitel beginnt mit den grundsätzlich erforderlichen Bestandteilen in Android, um Sensoren zu nutzen. Klassen, Schnittstellen und Methoden werden hierfür erklärt. In der Folge wird auf die Auswertung von Bewegung und auf die Ermittlung der Position eingegangen. Am Ende dieses Kapitels sollte ein Verständnis für die Nutzung von Sensoren für Navigationsaufgaben aus einer Android Implementationssicht in der Theorie vorhanden sein.

Android stellt im Rahmen des Package android.hardware alle Klassen und Interfaces bereit, um auf Sensoren zugreifen und Daten auslesen zu können. Die Sensoren werden vom SensorManager, einem zentralen Android-Dienst, verwaltet. Mit *getSystemService* kann eine Referenz geholt werden (vgl. Post /9/, Seite 272). Im nächsten Schritt muss eine Referenz auf den betreffenden Sensor bezogen werden. Dies erfolgt mit *getDefaultSensor*.

Das Auslesen erfolgt nicht mittels Lesebefehl, sondern es wird ein Listener aktiviert, der benachrichtigt wird, sobald sich der aktivierte Sensor verändert. Dies ist aus Effizienzgründen so konzipiert und geht so weit, dass die Sensoren nur dann genutzt werden, wenn dies etwa durch eine gerade sichtbare Anwendung, überhaupt benötigt wird. Hier kommen die zuvor erläuterten Stati von Android Anwendungen zum Tragen, die genutzt werden, um den Listener anzuhalten, zu reaktivieren oder gänzlich zu beenden.

Die wichtigste Methode für die weitere Verarbeitung ist *onSensorChanged*, die dann angesprungen wird, wenn neue Sensordaten geliefert werden.

3.3.1 Allgemeines zu Sensorik in Android

Android unterscheidet bei Sensoren folgende 3 Kategorien.

- Bewegungssensoren: Messen lineare und drehende Bewegungen in 3 Achsen
- Positionssensoren: Messen die relative Lage zur Erde
- Umgebungssensoren: Messen Umgebungsbedingungen wie Temperatur, Luftdruck, etc.

Für die weitere Strukturierung wird eine daran angelehnte Gruppierung gewählt, wobei nicht die Sensoren, sondern die Aufgabe im Mittelpunkt steht und praktikable Methoden in die Er-

läuterung der jeweiligen Sensoren einbezogen werden. Damit ergeben sich die Auswertung von „Bewegung“, „Position“ und der Vollständigkeit halber auch „andere Sensordaten“. Diese Struktur kann zu Überschneidungen führen, etwa dort, wo der Luftdruck als „andere Sensordaten“ ausgewertet wird, jedoch mit der entsprechenden Methode die Höhe vom Meeresspiegel errechnet wird. Was somit einer „Position“ entspricht. Dieser Umstand wird akzeptiert.

Tabelle 5: Liste von Sensoren in Android zeigt eine Liste aller Sensoren, die unter Android angesprochen werden können und zur Nutzung für Navigationszwecke sinnvoll sind.

Sensor	Type	Beschreibung	Einsatzgebiet
TYPE_ACCELEROMETER	Hardware	Misst die Beschleunigungskraft in m/s2, die auf das Smartphone auf allen 3 Achsen (x, y, z) wirkt. Dies beinhaltet auch die Erdanziehungskraft.	Bewegungserkennung, Schütteln, Tilt, etc.
TYPE_GRAVITY	Software oder Hardware	Misst die Erdanziehungskraft, die auf allen 3 Achsen (x, y, z) wirkt.	Bewegungserkennung, Schütteln, Tilt, etc.
TYPE_GYROSCOPE	Hardware	Misst die Drehrate eines Smartphone in rad/s um die 3 physischen Achsen (x, y, z).	Rotationserkennung
TYPE_LINEAR_ACCELERATION	Software oder Hardware	Misst die Beschleunigung, exkludiert dabei jedoch die Erdbeschleunigung	Beschleunigung entlang einer Achse
TYPE_MAGNETIC_FIELD	Hardware	Misst das geomagnetische Feld auf allen 3 Achsen (x, y, z) in µT.	Kompass
TYPE_ORIENTATION	Software	Misst den Grad der Drehung auf allen 3 Achsen (x, y, z). Ab API Level 3 kann die Neigungs- und Rotationsmatrix auch durch den Erdanziehungskraft und Geomagnetischen Feldsensor ermit-	Gibt Auskunft über die Lage

		telt werden	
TYPE_ROTATION_VECTOR	Software oder Hardware	Misst die Lage durch Angabe der Rotationsvektoren auf allen Achsen	Bewegungs- und Rotatonserkennung
TYPE_PRESSURE	Hardware	Misst den atmosphärischen Druck in hPa	Barometer, Wettervorhersage, Meereshöhe

Tabelle 5: Liste von Sensoren in Android (Quelle: Android)

Die in Tabelle 5: Liste von Sensoren in Android in der Spalten Type als „Hardware oder Software“ oder „Software“ ausgewiesenen Sensoren sind in Anlehnung an die bereits beschriebene Sensor Fusion als solche markiert. Diese werden auch virtuelle Sensoren genannt und nutzen in der Regel mehrere Hardwaresensoren, um zu möglichst zuverlässigen Ergebnissen zu gelangen (vgl. Meier /7/, Seite 484).

Android stellt im Rahmen des Package android.hardware alle Klassen und Interfaces für die Sensornutzung bereit. Diese werden in weiterer Folge erklärt und ihre konkrete Nutzung anhand von Beispielen illustriert.

Sensormanager. Diese Klasse dient zum Erstellen einer Instanz des Sensor Service. Darin werden unterschiedlichste Methoden zum Auslesen von Sensoren angeboten. Darüber hinaus bietet die Klasse Konstanten, die verwendet werden können, die Genauigkeit der Messung auszulesen bzw. diese zu bestimmen.

Sensor. Diese Klasse dient ebenfalls zum Erstellen einer Instanz eines bestimmten Sensors und bietet unter anderem die Möglichkeit die Eigenschaften von Sensoren auszulesen.

Sensor Event. Diese Klasse erstellt ein sogenanntes Sensor Event Object und bietet Informationen bei Auftreten eines Ereignisses. Das Objekt enthält folgende Informationen: Der Sensor, der ein Ereignis ausgelöst hat (mehr dazu unter SensorEventListener), die Genauigkeit der ausgelesenen Daten und der Zeitstempel eines Ereignisses.

SensorEventListener. Mit diesem Interface werden 2 Methoden zum Auslesen des Sensors auf Basis eines Ereignisses festgelegt. Es wird ein Ereignis ausgelöst, wenn sich der Wert eines Sensors ändert oder wenn sich seine Genauigkeit ändert.

3.3.2 Methoden für Sensorik in Android

In diesem Kapitel werden die wichtigsten Methoden zur Verwendung von Sensoren dargestellt. Diese dienen zum richtigen Lesen, aber darüber hinaus auch zur weiteren Bearbeitung

von Sensor Rohdaten. Die meisten Public Methods werden hier im Anschluss erklärt. Einige werden erst im Zusammenhang mit den entsprechenden Klassen und Schnittstellen angeführt. Dies erscheint verständlicher.

Bevor mit den weiteren Kapiteln fortgefahren wird, noch ein paar Tipps zur Beachtung:

- Das Auslesen der Bewegungssensoren ist stromintensiv. Zur Schonung der Akkus ist daher empfohlen, bei der Änderung des Ablauf-Status, etwa beim Wechsel in Paused oder Stopped, das Abfragen der Sensoren wieder auszuschalten. Dies kann mit einem Unregister erfolgen.
- Testen mit einem Emulator ist aufgrund mangelnder Sensordaten nur eingeschränkt möglich. Nur das grundsätzliche Look-and-Feel kann abgeleitet werden, aber es können keine Sensordaten gemessen werden. Ausgenommen davon ist der zuvor erwähnte Sensor-Simulator.
- In der Ablauflogik muss darauf geachtet werden, dass nicht innerhalb der onChanged Funktion zu viel Logik eingebaut wird. OnChanged wird immer angesprochen, wenn der Sensor eine Änderung misst. Wird die onChanged Funktion zu umfangreich, kann dies zu Problemen führen, da die Kontrolle nicht rechtzeitig an das Betriebssystem zurückgegeben werden kann während der Sensor weiterhin Daten senden möchte.

3.3.2.1 OnSensorChanged

Wird vom Listener aufgerufen, wenn sich ein Ergebniswert einer der registrierten Sensoren ändert. OnSensorChanged wurde bereits zuvor erwähnt. Es handelt sich um eine zentrale Methode, die genutzt wird, wenn sich Sensordaten ändern und die Verarbeitungslogik bei Sensordatenänderung beinhaltet.

*Public abstract void **onSensorChanged**(**SensorEvent** <event>)*

3.3.2.2 getSensorList

Mit dieser Methode können die verfügbaren Sensoren ausgelesen werden. Dies kann gezielt unter Angabe des Sensortypes erfolgen oder für alle. Entwickelt man Anwendungen für unterschiedlichste Android Smartphones so ist nicht vorhersehbar, welche Sensoren verfügbar sind. In der Regel wird eine Anwendung daher damit starten, dass eine Prüfung der Sensoren auf Vorhandensein vorgenommen wird.

*Public List<Sensor> **getSensorList**(int type)*

3.3.2.3 OnAccuracyChanged

Wird vom Listener aufgerufen, wenn sich die Genauigkeit des Sensors von sich aus ändert.

*Public abstract void **onAccuracyChanged**(**Sensor** <sensor>, int accuracy)*

3.3.2.4 registerListener

Diese Methode ermöglicht die Registrierung eines bestimmten Sensors. Damit wird festgelegt, dass auf die Ereignisse von angegebenen Sensoren reagiert wird. Die abgefragte Datenqualität kann hier ebenfalls angegeben werden. Diese Methode ist dann besonders wichtig, wenn die Anwendung wieder reaktiviert wird.

*Public bool **registerListener**(**SensorEventListener** <listener>, **Sensor** <sensor>, rate)*

Als konstante Raten können bei *rate* diese Angaben gemacht werden:

- SENSOR_DELAY_NORMAL
- SENSOR_DELAY_UI
- SENSOR_DELAY_GAME
- SENSOR_DELAY_FASTEST

3.3.2.5 unregisterListener

Auflösung der Registrierung für einen Sensor. Damit wird das Reagieren auf einen Sensor beendet. Zur Schonung der Systemressourcen sollten nur jene Sensoren registriert sein, die für die Verarbeitung relevant sind und in Phasen, wo keine Daten benötigt werden, deaktiviert werden.

*Public void **unregisterListener**(**SensorEventListener** <listener>, **Sensor** <sensor>)*

3.3.3 Auswerten von Bewegung in Android

In diesem Unterkapitel werden alle Klassen, Interfaces und Methoden beschrieben, die Bewegung messen oder umwandeln oder für Navigationsaufgaben interessant sind.

3.3.3.1 Type_Accelerometer

Mit diesem Interface wird die rohe Beschleunigung, die auf das Android Smartphone wirkt, gemessen. Es werden alle 3 Achsen berücksichtigt und ebenso die Beschleunigung der Erdanziehung von 9,81 m/s^2. Es wird der hardwaremäßige Beschleunigungsensor direkt ausgewertet.

Tabelle 6: Accelerometer erklärt die erhaltenen Messwerte.

Sensor	Sensor event data	Description	Units of measure
TYPE_ACCELEROMETER	SensorEvent.values[0]	Acceleration force along the x axis (including gravity).	m/s2

	SensorEvent.values[1]	Acceleration force along the y axis (including gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (including gravity).	

Tabelle 6: Accelerometer (Quelle: Android)

3.3.3.2 Type_Gravity

Bei diesem Sensor handelt es sich um einen virtuellen Sensor, der auf dem hardwaremäßigen Beschleunigungssensor aufbaut. Komatineni /5/ beschreibt diesen auf Seite 857. Bereits in einem der der vorangegangenen Kapitel wurde Sensor Fusion erläutert. Der Gravity Sensor stellt eine praktische Implementierung dar.

Er nutzt auch den Gyroscope Sensor und Magnetic Field Sensor zur Berechnung der Gravitation ausgehend von der Lage in der sich das Android Smartphone befindet.

Sensor	Sensor event data	Description	Units of measure
TYPE_GRAVITY	SensorEvent.values[0]	Force of gravity along the x axis.	m/s2
	SensorEvent.values[1]	Force of gravity along the y axis.	
	SensorEvent.values[2]	Force of gravity along the z axis.	

Tabelle 7: Gravity (Quelle: Android)

Im ruhigen Zustand liefert der Gravitationssensor die gleichen Werte wie der Beschleunigungssensor.

3.3.3.3 Type_Linear_Acceleration

Bei diesem Sensor handelt es sich ebenso wie beim Gravitationssensor, um einen virtuellen Sensor, der seine Werte vom hardwaremäßigen Beschleunigungssensor ableitet, wobei die Gravitation herausgerechnet wird. Für den Einsatz von Beschleunigungssensoren für Trägheitsnavigation ist dieser virtuelle Sensor daher sehr nützlich.

Sensor	Sensor event data	Description	Units of measure

TYPE_LINEAR_ACCELERATION	SensorEvent.values[0]	Acceleration force along the x axis (excluding gravity).	m/s2
	SensorEvent.values[1]	Acceleration force along the y axis (excluding gravity).	
	SensorEvent.values[2]	Acceleration force along the z axis (excluding gravity).	

Tabelle 8: Linear Acceleration (Quelle: Android)

Die Syntax ist analog zu Type_Accelerometer und Type_Gravity. Es muss lediglich der Sensortyp Type_Linear_Acceleration eingesetzt werden.

3.3.3.4 Type_Gyroscope

Das standardmäßige Type_Gyroscope liefert rohe und ungefilterte Daten der Drehrate auf allen Achsen.

Sensor	Sensor event data	Description	Units of measure
TYPE_GYROSCOPE	SensorEvent.values[0]	Rate of rotation around the x axis.	rad/s
	SensorEvent.values[1]	Rate of rotation around the y axis.	
	SensorEvent.values[2]	Rate of rotation around the z axis.	

Tabelle 9: Gyroscope (Quelle: Android)

3.3.4 Messen der Position in Android

3.3.4.1 Type_Rotation_Vector

Der Rotations Vector Sensor liefert die Rotationsvektoren entlang der 3 Achsen.

Der Rotationsvektor repräsentiert die Orientierung des Gerätes als eine Kombination aus Winkel und einer Achse, um welche eine Drehung stattfindet. Die Elemente eines Rotationsvektors sind einheitslos. Die Achsen, x, y und z, sind gleich wie beim Beschleunigungssensor definiert. Das Referenzkoordinatensystem ist als direkt orthonormal definiert: X ist demnach das Vektorprodukt von Y und Z, ist tangential zur Erde und zeigt nach Osten.

Y ist tangential zur Erde und zeigt nach Norden. Z zeigt zum Himmel und ist senkrecht zur Erde.

Sensor	Sensor event data	Description	Units of measure
TYPE_ROTATION_VECTOR	SensorEvent.values[0]	Rotation vector component along the x axis (x * sin(θ/2)).	Unitless
	SensorEvent.values[1]	Rotation vector component along the y axis (y * sin(θ/2)).	
	SensorEvent.values[2]	Rotation vector component along the z axis (z * sin(θ/2)).	
	SensorEvent.values[3]	Scalar component of the rotation vector ((cos(θ/2)).1	

Tabelle 10: Rotation Vector (Quelle: Android)

Ausgehend vom Rotationsvektor kann mittels geeigneter Methoden eine verwertbare Umrechnung stattfinden. Es wird damit die Position ermittelt.

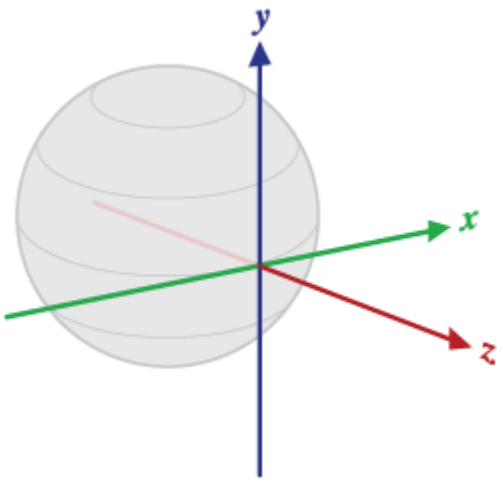

Abbildung 16: Koordinatensystem (Quelle Android)

Android bietet Methoden für die Umrechnung. Die Methode *getQuaternionFromVector()* ermöglicht die Umrechnung in ein normalisiertes Quaternion. Und *getRotationMatrixFromVector()* konvertiert ein Rotations Vector Sensor Ergebnis in eine Rotationsmatrix. Dies kann mit *getOrientation()* weiterverwertet werden. Das Ergebnis entspricht dem von Type_Orientation.

3.3.4.2 Type_Magnetic_Field

Der Magnetische Feldsensor wird für die Implementierung eines Kompasses herangezogen, er misst auch das Erdmagnetfeld.

Sensor	Sensor event data	Description	Units of measure
TYPE_MAGNETIC_FIELD	SensorEvent.values[0]	Geomagnetic field strength along the x axis.	µT
	SensorEvent.values[1]	Geomagnetic field strength along the y axis.	
	SensorEvent.values[2]	Geomagnetic field strength along the z axis.	

Tabelle 11: Magnetic Field (Quelle: Android)

Type_Magnetic_Field liefert rohe magnetische Feldstärkedaten entlang der 3 Achsen und wird verwendet, wenn ein Kompass programmiert werden soll. Allerdings geschieht dies mittels virtuellem Sensor, der die Daten mit anderen Sensordaten kombiniert und weiter verarbeitet.

3.3.4.3 getRotationMatrix

Errechnet die Neigungsmatrix I, sowie die Rotationsmatrix R basierend auf der Erdanziehungskraft und dem geomagnetischen Feld.

*Public static boolean **getRotationMatrix** (float[0-2] R, float[0-2] I, float[0-2] gravity, float[0-2] geomagnetic)*

- Value[0]: Azimuth, Rotation um die z-Achse (zeigt nach Osten)
- Value[1]: Pitch, Rotation um die x-Achse (zeigt nach Süden)
- Value[2]: Roll, Rotation um die y-Achse

Achtung, es wird ein anderes Koordinatensystem, als bei der *getOrientation* verwendet.

3.3.4.4 Type_Orientation

Aufgrund der Tatsache, dass Type_Orientation ausgelaufen ist und nicht mehr verwendet werden soll oder kann, erfolgt hier keine weitere Erläuterung dazu.

3.3.4.5 getOrientation

Errechnet die Lage des Mobilen Endgerätes basierend auf der Rotationsmatrix. Als Ergebnis werden folgende Werte ermittelt:

- Value[0]: Azimuth, Rotation um die z-Achse (zeigt nach Westen)
- Value[1]: Pitch, Rotation um die x-Achse (zeigt nach Norden)
- Value[2]: Roll, Rotation um die y-Achse

Achtung, es wird ein anderes Koordinatensystem, als bei der Rotationmatrix verwendet.

Public static float[] ***getOrientation*** *(float [] R, float[] values)*

Das Ergebnis erfolgt in Radianten, positiv und in Uhrzeigerrichtung.

3.3.4.6 Type_Pressure

Dieser Sensor liefert den vorliegenden atmosphärischen Druck. Dieser kann für Anwendungen im Bereich von Wettervorhersagen und für die Höhenbestimmung genutzt werden. Zur Umwandlung der Daten in brauchbare Ergebnisse wird *getAltitude* verwendet.

Sensor	`Sensor event data`	Description	Units of measure
`TYPE_PRESSURE`	`event.values[0]`	Atmospheric pressure	hPa

Tabelle 12: Pressure (Quelle: Android)

3.3.4.7 getAltitude

Errechnet den Höhenunterschied aufgrund der Differenz des atmosphärischen Drucks. Als Ergebnis wird der Höhenunterschied von 2 Punkten in Meter angegeben.

Public static float ***getAltitude*** *(float1, float2)*

Als float1 wird der Referenzpunkt angegeben, üblicherweise der Druck auf Meeresniveau. Und float2 ist der Druck aus dem Sensor, der zuvor ausgelesen wurde.

4 Anwendungsbeispiel Navigation

Nachdem in den vorangegangenen Kapiteln das Grundlagenwissen, zu Bewegungssensoren und darauf aufbauend Navigationsaufgaben in einem Smartphone, aufgebaut wurde, sowie die Programmierung mittels Android zur Nutzung dieser erklärt wurde, wird in diesem Kapitel der praktische Zugang vermittelt.

Möchte man „echte“ Trägheitsnavigation realisieren, so ist, wie bereits in den Kapiteln davor beschrieben, die Anwendung des Strapdown-Algorithmus eine Voraussetzung. Das bedeutet, dass die Positionsänderung aus doppelter Integration der Beschleunigung und einfacher Integration der Drehgeschwindigkeit ermittelt wird. Dazu muss das Messgerät mit dem zu messenden Objekt fest verbunden sein und jede Bewegung mit dem Objekt mitausgeführt werden.

Die Möglichkeit der fixierten Kopplung von Messgerät zu Messobjekt ist bei Flugzeugen und Schiffen gegeben. Möchte man jedoch die Fortbewegung eines Menschen messen und die Messung über ein lose gekoppeltes Smartphone durchführen, so stößt man auf Hindernisse, weil die Lage des Smartphones zum Messobjekt (dem Menschen) nicht messbar ist und damit ein essentieller Parameter fehlt. Weiters erzeugt der menschliche Gang durch die Erschütterungen erhebliche, kurzzeitige Beschleunigungen, welche die Ausschläge anderer Beschleunigungswerte überlagern und damit eine richtige Interpretation der Sensordaten erschweren.

Folgende Aufzeichnung, Abbildung 17: Beschleunigung beim Gehen, zeigt die auftretenden Beschleunigungen beim Gang eines Menschen über eine Distanz von etwa 5 Metern. Beachtet man die Z-Achse, die die auftretende Beschleunigung in der Vertikale darstellt, so erkennt man die regelmäßigen Ausschläge, die durch die Schritte hervorgerufen werden. Deren Amplituden sind deutlich stärker als jene Beschleunigungen nach Vorne und zur Seite. Da sich die Messung immer auf das Smartphone bezieht, wird eine leichte Seitwärts-

oder Vorwärtslage eine Beschleunigung auch auf X- und Y-Achse erkennen lassen. Ein Rückschluss aus der gemessenen Beschleunigung auf die zurückgelegte Wegstrecke wird dadurch wesentlich verfälscht.

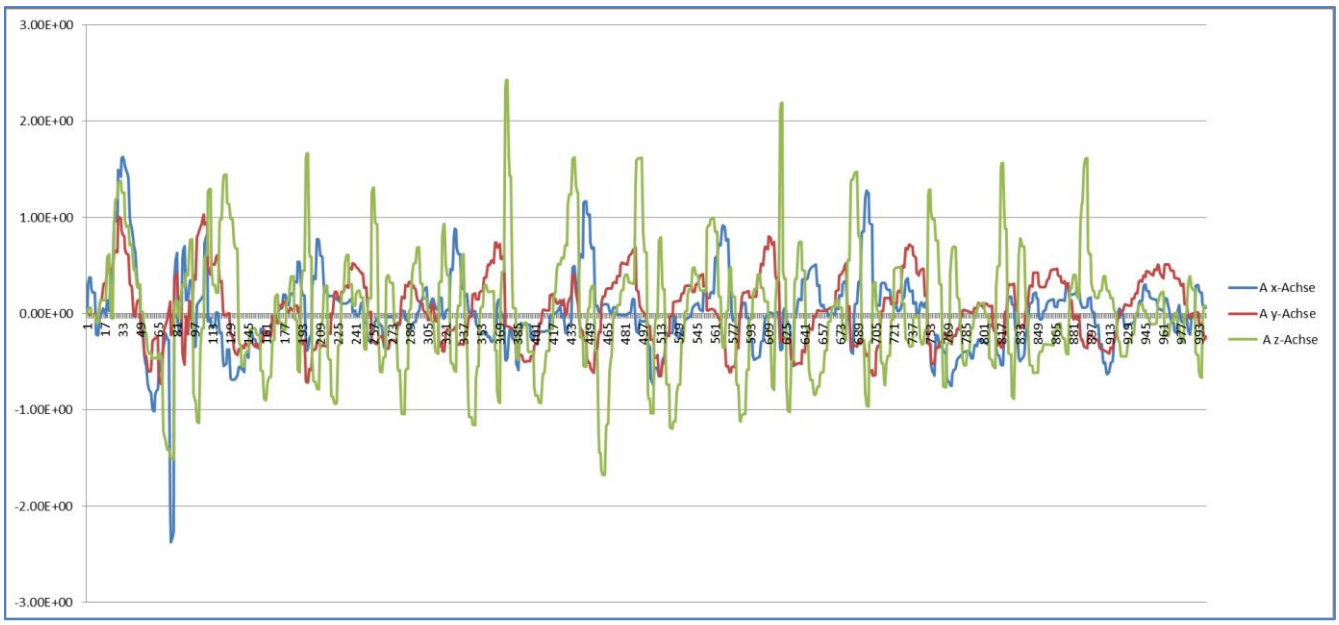

Abbildung 17: Beschleunigung beim Gehen

Es ist absehbar, dass hier eine Alternative gefunden werden muss, um eine verwendbare Lösung zu erhalten. Eine Trägheitsnavigation im klassischen Sinn wird nicht funktionieren. Dennoch ist es möglich mittels Bewegungssensoren und Lagesensoren Navigationsaufgaben zu übernehmen. Ob und in welcher Qualität dies möglich ist, wird die weitere Ausarbeitung behandeln. Die dargestellten Messwerte wurden durch ein eigenes Experiment erhoben. Dazu wurde die erstellte App und dabei die implementierte Protokollierung genutzt. Die Datenaufbereitung wurde in Microsoft Excel vorgenommen.

4.1 Herangehensweise

Das menschliche Gehen erzeugt merkbare und gut messbare Beschleunigungen entlang der Z-Achse, also vertikal. Bei jedem Schritt ist ein Beschleunigungsmuster erkennbar, das aus einem außergewöhnlichen Ausschlag in eine Richtung danach in die andere Richtung besteht. Dieses Muster kann als Schritt detektiert und gezählt werden. Geht man davon aus, dass die Schrittlänge relativ konstant ist, so wird dadurch eine Distanzmessung möglich. Die Einheit lautet dann entsprechend: „Schritt“.

In den durchgeführten Tests wurden Schwellenwerte zwischen 1 bis 1,5 m/s2 eingestellt, was zu brauchbaren Resultaten geführt hat. Die Messung wird von Post /9/ ab Seite 292 beschrieben und auch die empfohlenen Schwellwerte wurden daraus abgeleitet.

Die Richtung ist schwieriger zu ermitteln. Dazu wurden 3 Methoden angedacht und zu experimentalen Zwecken ausgearbeitet. Die ursprüngliche Methode sollte aus der Beschleunigung die Bewegungsrichtung ableiten. Dazu würden die Beschleunigungen auf X- und Y-

Achse in eine Distanz umgerechnet. Die Werte würden bezogen auf die Gerätelage kumuliert und daraus ein Richtungsvektor ermittelt. Der Winkel des Richtungsvektors gäbe die Gehrichtung wieder. Diese Methode war im theoretischen Modell plausibel, die Messergebnisse allerdings nicht brauchbar. Deshalb auch der Konjunktiv, eine Implementierung nach dieser Methode wurde nicht weiter verfolgt. Bei den weiteren Methoden wurde der Kompass genutzt, der aus der Orientierung abgeleitet wird. Also dem Ergebnis aus einem virtuellen Sensor, der die Orientierung bezogen auf die Erde emuliert und dafür die Messergebnisse von anderen Sensoren (Beschleunigung, Drehraten, elektromagnetisches Feld) verwendet.

Ebenso wurde das Gyroskop, mit dessen Hilfe die Drehbewegungen ausgehend von einem Ausgangspunkt aufgezeichnet werden können, näher untersucht. Letztere Methode hat sich bei den Tests als bevorzugte Methode herausgestellt, da hier die Latenzzeiten zwischen Bewegung und Erkennen dieser am günstigsten waren.

Wichtig ist dabei, dass das Geräte in die Gehrichtung zeigt, denn es wird die Lage als Gehrichtung interpretiert. Das ist eine wesentliche Rahmenbedingung. Ebenso muss das Smartphone waagerecht gehalten werden. Die Voraussetzung gilt für beide in Betracht kommende Methoden.

Ein Koordinatensystem ist in jedem Fall erforderlich. Es muss ein geräteunabhängiges System vorhanden sein, in das alle Bewegungen übertragen werden. In diesem Beispiel wird ein sehr einfaches genutzt, das nur aus den beiden Bewegungsrichtungen, vor und zurück sowie links und rechts, besteht. Im Zuge der Implementierung wurde mit relativer Bewegungsaufzeichnung experimentiert. Dies erfordert jedoch weit mehr Rechenaufwand und birgt daher mehr Ungenauigkeitsprobleme. Der ursprüngliche Ansatz auf ein Koordinatensystem zu verzichten, musste daher fallen gelassen werden.

Die wichtigen Passagen werden im Zuge der Erläuterungen zum Coding beschrieben. Es hat sich gezeigt, dass die schwierigste Aufgabe darin besteht, die Messergebnisse in ein Koordinatensystem zu übertragen und konsistente Navigationsangaben zu erzeugen. Dafür sind unter anderem Winkelfunktionen und Dreiecksfunktionen (Pythagoras) erforderlich.

Die Nutzung des Gyroskops stellte sich erst während der erfolglosen Versuche der anderen Methoden als günstigste heraus. Zur Ermittlung des Drehwinkels aus der dem Integral der Drehrate wurde folgende Untersuchung angestellt. Siehe dazu Abbildung 18: Auswertung des Gyroskops. Es wurde das Integral gebildet und als Parameter Fläche dargestellt. Zum Vergleich wurde auch der Kompasswinkel in das Diagramm aufgenommen. Dabei wurden, zur besseren Darstellung, die Fläche durch 3 und der Kompasswinkel durch 10 dividiert.

Die Messung wurde gestartet und das Smartphone flach liegend 10 Mal um die Z-Achse gedreht. Aus der entstandenen Fläche wurde die Konstante ermittelt, die angibt wie viel Fläche ein Grad ergibt. Somit wurde -0.01589698 ermittelt, wenn eine Drehung um ein Grad nach rechts vollzogen wird. Ebendiese Konstante wurde auch für bei der programmtechnischen Umsetzung zur Umrechnung genutzt.

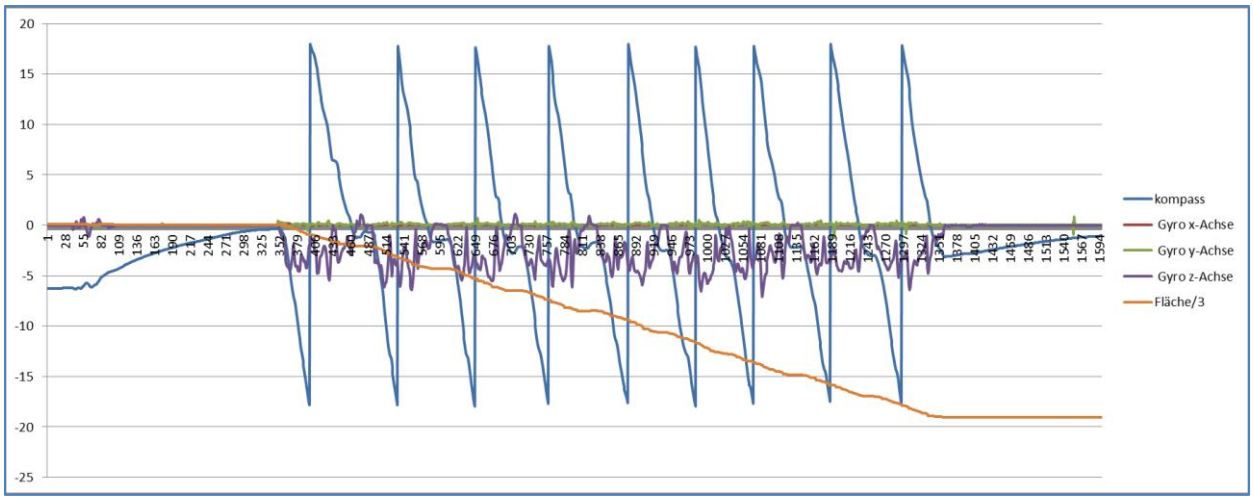

Abbildung 18: Auswertung des Gyroskops

4.2 Anwendungsbeispiel „Guide me home“

Mit „Guide me home“ soll eine sinnvolle Anwendung vorgestellt werden und ein Prototyp entwickelt werden. Die Anwendung soll gemachte Schritte zählen und dabei die Gehrichtung von einem Ausgangspunkt aufzeichnen und den Benutzer wieder zurückführen. Eine Activity könnte wie in Abbildung 19: Display Guide me home aussehen.

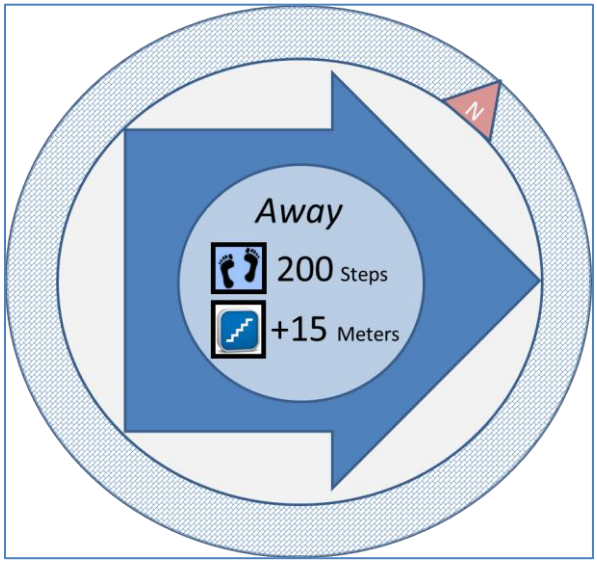

Abbildung 19: Display Guide me home

Anwendungsgebiete können überall dort auftreten, wo GPS Signale nicht erreichbar sind. Etwa in einer Tiefgarage. Man verlässt das Auto, denkt natürlich nicht daran sich die Farbe des Parkdecks zu merken, und schon gar nicht die Parkplatznummer. Sofern man einen guten Orientierungssinn hat und sein Fahrzeug wieder findet, ist das kein Problem. Aber wenn nicht, so könnte die Einkaufstour oder die Rückkehr von einer Dienstreise in der Flughafengarage etwas länger dauern.

GPS als Navigationssystem funktioniert nur bei verfügbarem Satellitensignal. Ein bestimmtes Maß an Bewegung kann interpoliert werden, aber grundsätzlich gilt, ohne Signal keine Navigation.

Die wichtigsten Anforderungen und gewünschten Eigenschaften von „Guide me home“

- Es werden die zurück gelegten Schritte gezählt.
- Es können während dem Gehen mehrfach Richtungswechsel stattfinden, die zu einer Gesamtrichtung zusammengeführt werden, sodass eine Richtungsanzeige zum Ausgangspunkt möglich ist.
- Eine Rückführung zum Ausgangspunkt soll aktiviert werden, wenn eine erhebliche Richtungsänderung stattfindet, etwas mehr als 100° zum letzten Schritt
- Eine relative Schrittanzahl, die Schrittanzahl auf dem kürzesten Weg, ist wünschenswert
- Die Anzeige soll die Messwerte liefern, aber vor allem eine Richtungsanzeige ermöglichen
- Das Gerät muss vor dem Körper flach gehalten werden und in Gehrichtung zeigen.
- Eine Höhenveränderung wird mittels Barometer abgefragt.
- Möchte man im Sinne des Tiefgaragenbeispiels reüssieren, so sollte ein Fußmarsch von fünf Minuten zu einer Genauigkeit der Rückführung von maximal 50 Meter Abweichung vom Ausgangspunkt erzielt werden, etwa so weit, wie die Fernbedienung des Fahrzeugs wirkt oder das eigene Auto erkennbar ist.

4.3 Prototyp „Guide me home“

4.3.1 Kurzbeschreibung

Das zuvor beschriebene Anwendungsbeispiel wurde als Prototyp implementiert und auch einigen Tests unterzogen. Auf Funktionen den Bedienungskomfort betreffend wurde verzichtet und auch das Layout wurde funktional und zweckmäßig umgesetzt. Auf die Höhenmessung wurde mangels Luftdrucksensor verzichtet. Im Gegenzug wurden Sensorrohdaten während der Verarbeitung angezeigt und eine Protokollierung der Messwerte für eine anschließende Analyse weggeschrieben.

In diesem Kapitel werden die wesentlichen Aspekte der Umsetzung beschrieben. Der Sourcecode ist im Anhang enthalten und wurde auf dem in der Einleitung beschriebenen Smartphone entwickelt und getestet.

In der vorgestellten App werden zunächst alle erforderlichen Sensoren initialisiert. Das sind der Beschleunigungssensor und der Gyroskop Sensor. Weiters der Rotationsvektor Sensor, der für die Bestimmung des magnetischen Nordens verwendet wird. Beschleunigungen entlang der z-Achse, demnach senkrecht verlaufen, werden zur Schrittinterpretation herangezogen. Zwischen den Schritten werden alle Drehbewegungen des Gyroskops aufgezeichnet und daraus eine Schrittrichtung ermittelt. Wenn der Schritt abgeschlossen wird, werden die

Ergebnisse kumuliert und eine neue Gesamtrichtung bestimmt. Das Ergebnis wird sogleich um 180° gedreht, um Richtung zum Ausgangspunkt wiederzugeben.

Die implementierte Lösung verfügt über 3 Activities (Abbildung 20: Android Hierarchie), wobei die NavigateActivity.java das Kernstück darstellt. RichtungView.java dient zur Anzeige der Richtungspfeile und MovingAverage.java realisiert einen Filter, der Sensorrauschen durch ein gleitendes Durchschnittsverfahren eliminiert. Für die Anzeige der Richtungspfeile wurde eine Anleitung bei Post /9/, Seite 274 und für den Moving Average bei Milette /8/, Seite 108 genommen.

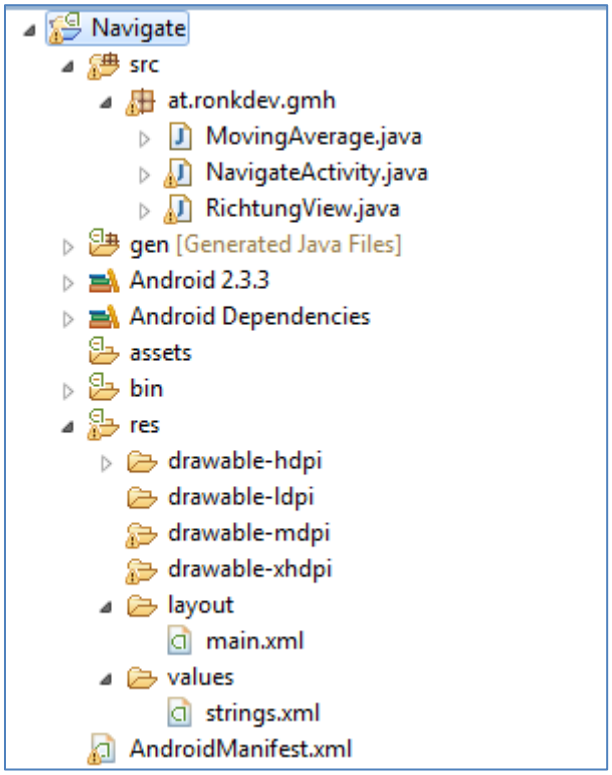

Abbildung 20: Android Hierarchie

Darüber hinaus sind eine Reihe von Includes eingefügt, die unterschiedliche Aufgaben realisieren. Siehe dazu Abbildung 21: Verwendete Includes.

```
package at.ronkdev.gmh;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;

import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.widget.TextView;
```

Abbildung 21: Verwendete Includes

Das Auslesen der Sensordaten hat sich als relativ einfach erwiesen. Schwieriger ist die Umrechnung der Sensorwerte in nutzbare Ergebnisse. Das Display ist wie folgt aufgebaut.

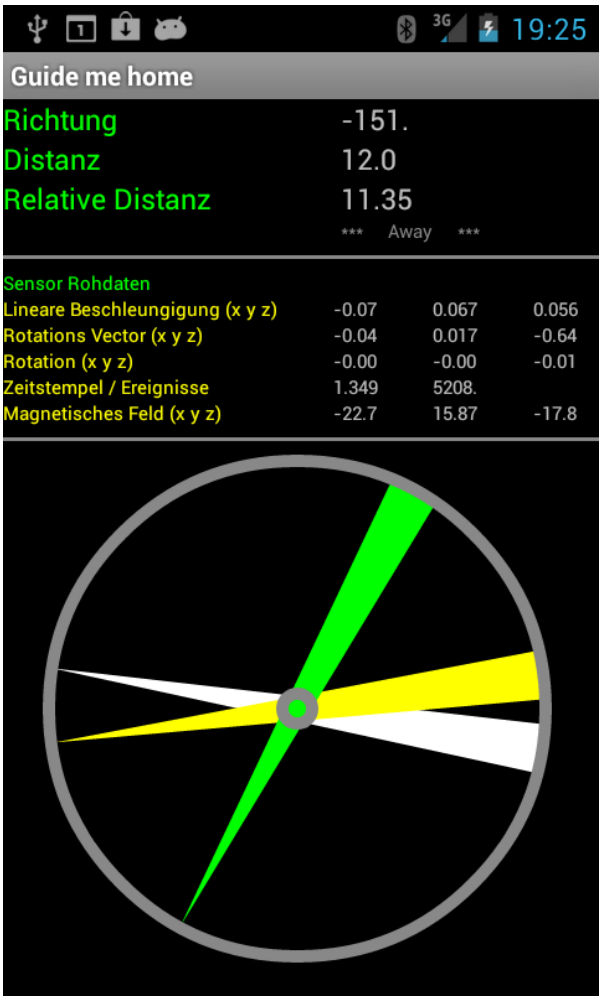

Abbildung 22: Anzeige Layout

Abbildung 22: Anzeige Layout zeigt den Aufbau des Displays. Im oberen Teil werden die primären Ergebnisse angezeigt. Die Richtung, die Distanz und die relative Distanz.

Die Richtung wird jeweils in Grad von Norden angegeben im Bereich von -180 bis +180. Die Distanz wird in Schritten angegeben. Die Relative Distanz wird aus dem Vektor ermittelt, der sich aus den kumulierten Schritten nach vor und zur Seite aus der Sicht des Koordinatensystems ergeben.

Unter Sensor-Rohdaten sind die Daten, die der Sensor beim letzten Event geliefert hat, angezeigt.

Drei Richtungsanzeiger werden optisch dargestellt. Grün zeigt in die Richtung des Ausgangspunktes und ist somit der interessanteste Zeiger. Weiß gibt den magnetischen Norden an. Dem gelben Zeiger ist derzeit kein Wert zugeordnet und so bleibt dieser auf 0 stehen.

In der Routine updateGui werden die Feldinhalte für die Ausgabe auf dem Display aktualisiert. Die Vielzahl von Ausgabefeldern gibt dem Studierenden oder Interessierten die Möglichkeit die angezeigten Inhalte zu variieren, um etwa Zwischenwerte anzuzeigen oder andere Werte, die von Interesse sind. Eine einfache Adaptierung dieser Routine bietet entsprechende Möglichkeiten, eigene Untersuchungen auf dieser Grundlage aufzubauen.

Die Logik ist so aufgebaut, dass bei Auftreten von Sensordaten geprüft wird, welcher Sensor das Ereignis ausgelöst hat. Im Prototyp werden der lineare Beschleunigungssensor, der das Gyroskop und auch der Rotationsvektorsensor genützt. Es werden weitere Sensoren ausgelesen, jedoch nur zu Informationszwecken.

Werden verwertbare Sensordaten erkannt, werden Routinen angesprungen, die eine weitere Verarbeitung vornehmen.

Bei den angeführten Beispielen wurde aus Formatierungsgründen bewusst auf die gesamte Breite der Codezeilen verzichtet. Der gesamte Code befindet sich im Anhang, die hier angeführten Beispiele sind als erläuternder Auszug zu betrachten. Die nicht erfolgreichen Ansätze zur Navigation wurden im Sourcecode belassen, jedoch auskommentiert. Als V2 markierte Passagen beziehen sich auf eine Lösung, die die Zeigerichtung des Smartphones mittels Kompass auswertet. Wie eingangs beschrieben erschien die Latenzzeit des Gyroskops für die Anwendung brauchbarer als jene des Rotationsvektorsensors. Eigentlich unterscheiden sich die Varianten nur marginal. V3 markiert jene Passagen wo eine Lösung mit der Beschleunigung getestet wurde. Die Abweichung ist hierbei erheblicher und daher finden sich auch mehr auskommentierte Stellen mit Bezug zu V3.

Es wurde intensiver Gebrauch von den Debugging Methoden von Eclipse gemacht, die ein rasches Validieren und Testen von Programmänderungen ermöglichen.

4.3.2 Implementierung: Sensoren

Um die Sensoren ansprechen zu können, wird eine Referenz auf den erforderlichen Dienst geholt. Dies geschieht in onCreate mit der in Abbildung 23: Syntax Sensormanager angeführten Syntax.

```
//          Sensoren: Der Sensormanager wird aktiviert

    mSensorManager = (SensorManager) this.getSystemService(Context.SENSOR_SERVICE);
```

Abbildung 23: Syntax Sensormanager

Die Nutzung der Sensoren erfolgt in der Form, dass ein Listener aktiviert wird, der den Programmablauf bei Auftreten eines Sensorevents in den entsprechenden Programmteil führt. Mit der in Abbildung 24: Syntax Sensor Listener gezeigten Methode wird Android mitgeteilt, bei welchen Sensorevents eine Reaktion ausgelöst werden soll. In diesem Fall werden der lineare Beschleunigungssensor, der Rotationsvektorsensor, der magnetische Feldsensor und der Gyroskop Sensor abonniert. In der aktuell genutzten Variante werden der lineare Beschleunigungssensor und der Gyroskop Sensor genutzt. Alle werden als Sensor Rohdaten auf dem Display angezeigt.

```
//          Sensoren: Die erforderlichen, gewünschten Sensoren werden für die Verwe

        mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensc
        mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensc
        mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensc
        mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensc
        refreshing=true;
```

Abbildung 24: Syntax Sensor Listener

Der Aufruf geschieht in OnResume. Es könnte in unserem Fall auch in OnCreate aufgerufen werden, da die unregister Methode (siehe Abbildung 25: Syntax Senoren deaktivieren) erst in onDestroy aufgerufen wird. Und somit die ganze Zeit Sensordaten empfangen werden, auch dann, wenn die Activity nicht im Vordergrund ist. Das ist für diese Anwendung erforderlich. In anderen Fällen, etwa bei Spielen, die die Beschleunigungssensoren für die Steuerung nutzen, kann bei onPause ein unregister der Sensoren durchgeführt werden. Dies schont in jedem Fall den Akkuverbrauch.

```
    protected void onDestroy() {
//          Sensoren: Erst hier werden die Sensoren un-registriert...

        mSensorManager.unregisterListener(this);
        refreshing=false;
```

Abbildung 25: Syntax Senoren deaktivieren

Möchte man mit Sensoren arbeiten so ist die entsprechende Activitiy mittels implements-Klausel SensorEventListener zu definieren (Abbildung 26: Syntax Definition der Klasse). Was zur Folge hat, dass die Methoden OnSensorChanged und OnAccuracyChanged zwingend definiert werden müssen.

```
public class NavigateActivity extends Activity implements SensorEventListener, Runn
```

Abbildung 26: Syntax Definition der Klasse

Implements Runnable wird angeführt, um die periodische Aktualisierung des Displays zu realisieren. Hierfür wurde ein Thread erzeugt, der eine Wartezeit vergehen lässt und danach die Aktualisierung veranlasst. Dies hat aber nichts mit den Sensoren zu tun.

OnAccuracyChanged wird aufgrufen, wenn sich die Genauigkeit des Sensors ändert. In der Regel braucht man diese Methode allerdings nur anzulegen ohne sie mit Logik zu füllen. Hingegen stellt die OnChanged Methode das Herzstück jeder Sensor-Applikation dar.

In Abbildung 27: Syntax onChanged Abschnitt ist die Verzweigungslogik für diesen Anwendungsfall dargestellt. Bei einigen Sensoren wird dazu eine Routine aufgrufen, die die gesamte Logik in sich birgt, während bei den anderen Sensoren einige Verarbeitungsschritte direkt in der Case Abfrage erfolgen.

```
    public void onSensorChanged(SensorEvent event) {

//          Sensoren: Das Kernstück. Sobald ein Sensoren-Ereignis auftritt, wird dies
//          angesprochen. Darin wird abgefragt, welcher Sensor das Ereignis ausgelöst
//          entsprechenden Verarbeitungsschritte durchgeführt.

//          Für die Navigationsaufgaben werden der Lineare Beschleunigungssensor, Gyr
//          Rotationsvektorsensor genutzt. Die anderen liefern lediglich Sensor-Rohda
//          zur Information.

        if (LetzteZeit==0) LetzteZeit = System.currentTimeMillis();

        switch (event.sensor.getType()) {
            case Sensor.TYPE_LINEAR_ACCELERATION:
                    do_acc_calc(event.values);
                break;
            case Sensor.TYPE_ROTATION_VECTOR:
                    do_rotvect_calc(event.values);
                break;
            case Sensor.TYPE_MAGNETIC_FIELD:
                    magfwertv1=event.values[0];
                    magfwertv2=event.values[1];
                    magfwertv3=event.values[2];
                break;
            case Sensor.TYPE_GYROSCOPE:

                ...

                break;
        }
    }
```

Abbildung 27: Syntax onChanged Abschnitt

Die Verarbeitung von Sensordaten des Gyroskops wird direkt in der Case Anweisung vorgenommen. Siehe dazu Abbildung 28: Syntax Bestimmen der Gehrichtung. Wie schon zuvor beschrieben basiert die primär implementierte Variante auf einer Richtungsbestimmung mit-

tels Gyroskop. Der Wert AktuelleLage wird für die weitere Verarbeitung, d.h. bei Erkennen eines Schrittes, als Richtung herangezogen.

```
            case Sensor.TYPE_GYROSCOPE:

//              Sensoren: Mit dem Gyroskop wird die Drehrate des Gerätes gemessen. D
//              sind Radianten je Zeit. Durch Integration kann der zurück gelegte Wi
//              und damit die Lageänderung.
//              Zur einfachen Ermittlung des Winkels wurde durch einen Versuch eine
//              Das Ergebnis wird in AktuelleLage abgelegt.

                    gyrowertv1=event.values[0];
                    gyrowertv2=event.values[1];
                    gyrowertv3=event.values[2];
                    if (offSet) {
                        AlteLage=AktuelleLage;
                        GLetzteZeit = System.currentTimeMillis();
                    }
                    GBewegungZurSeite = (float) (GBewegungZurSeite + (float) (event.
                    AktuelleLage = (-LageZuBeginn-(GBewegungZurSeite/(-0.015896989))
                    GLetzteZeit=System.currentTimeMillis();
            break;
```

Abbildung 28: Syntax Bestimmen der Gehrichtung

In Abbildung 29: Syntax Rotationsvektorsensor wird der Rotationsvektorsensor bei Auftreten eines Events ausgewertet. Man erhält 3 Werte, die Rotationswerte der 3 Achsen. Es handelt sich dabei um einen einheitslosen Vektor, der für die weiteren Berechnungen herangezogen wird. Mit den Methoden getRotationMatrixFromVector und getOrientation holt man sich die Lage des Smartphones. Die Lageangabe erfolgt wieder in 3 Parameter, wobei [0] die Drehung um die Z-Achse, also senkrecht, oder auch Azimuth, wiedergibt. Es wird daraus der Kompass abgeleitet und auch damit auch die Richtung, in die das Smartphone gerade zeigt. Der anfängliche Kompass-Wert wird auch für spätere Korrekturen als Konstante gespeichert, der für die Anzeige jeweils hinzugerechnet wird.

```
    private void do_rotvect_calc(float[] values) {

//          Sensoren: Mit dem Rotationsvektorsensor kann die Lage des Gerätes festeges
//          werden. Verwertbar ist hierbei der Azimuth-Wert aus getOrientation (Wert 0
//          der für die Kompassfunktion genutzt wird.
//          Der Rotationsvektorsensor ist ein virtueller Sensor, der aus anderen, phys
//          Sensoren ermittelt wird (Magnetischer Feldsensor, Gyroskope und Beschleuni
//V2Info    Der Rotationsvektorsensor wird primär für die einmalige Kalibrierung mit d
//V2Info    magnetischen Nordpol verwendet. In der Variante V2 wird auch die Lageänder
//V2Info    damit ausgewertet.

        rotvwertv1=values[0];
        rotvwertv2=values[1];
        rotvwertv3=values[2];

        mSensorManager.getRotationMatrixFromVector(RotationsMatrix, values);
        mSensorManager.getOrientation(RotationsMatrix, AusgleichsLage);
        KompassLage = Math.toDegrees(-AusgleichsLage[0]);
//V2     AktuelleLage = KompassLage
        if (offSet) {
            LageZuBeginn = -KompassLage;
        }
        }
```

Abbildung 29: Syntax Rotationsvektorsensor

In der Routine, in der die Beschleunigungsdaten weiter verarbeitet werden, befindet sich ein Abschnitt, der zur Qualitätsverbesserung einen permanenten Bias-Fehler erheblich reduzieren kann. Dieser Fehler äußert sich dadurch, dass die Sensoren auch Ergebnisse liefern, obwohl das Smartphone ruhig auf dem Tisch liegt. Es wurde beobachtet, dass diese Abweichung von Mal zu Mal variiert und daher nicht mit einem konstanten Wert ausgeglichen werden kann. Nachdem die Beschleunigungssensoren lediglich für relativ grobe Messungen herangezogen werden und nicht für die zurück gelegte Wegstrecke, wird die Verbesserung unerheblich sein. Gleiche Syntax könnte bei der Ermittlung des Offsets für das Gyroskop genutzt werden. Die Phase der Kalibrierung ist hier mit 5 Sekunden definiert und wird auf dem Display angezeigt.

```
//          Sensoren: Während der ersten x Millisekunden wird ein Offset berechnet, da
//          ruhig und flach liegen.

        if (offSet) {
            if ((startzeit + 5000) > System.currentTimeMillis()) {
                zaehl=zaehl+1;
                OffSet1Summe = (OffSet1Summe + values[0]);
                OffSet2Summe = (OffSet2Summe + values[1]);
                OffSet3Summe = (OffSet3Summe + values[2]);
                } else
                {
                offSet = false;
                OffSet1 = OffSet1Summe / zaehl;
                OffSet2 = OffSet2Summe / zaehl;
                OffSet3 = OffSet3Summe / zaehl;
                }
        }
```

Abbildung 30: Syntax Offset ermitteln

4.3.3 Implementierung: Navigation durch Richtung und Schrittzählung

Wir befinden uns nach wie vor in der Routine, die bei Auftreten eines Beschleunigungssensor Events aufgerufen wird. In Abbildung 31: Syntax Schrittzählung ist dargestellt, wie die Schrittzählung erfolgt. Dabei wird die vertikal auftretende Beschleunigung value[2] hinsichtlich Unter- oder Überschreitung eines definierten Schwellwertes gemessen und daraus ein erkannter Schritt interpretiert. Ist der Schritt unter dem Schwellwert wird der Schritt begonnen, ist er darüber wird er abgeschlossen. Bei Abschluss des Schritts, werden die Schrittdaten gesammelt und je nach Gehrichtung verarbeitet.

```
//          Navigation durch Richtung und Schrittzählung: Es wird die senkrechte Bes
//          auftritt, gemessen. Wird ein Schwellwert unterschritten wird ein Schritt
//          wird der Schritt beendet und alle notwendigen Berechnungen durchgeführt.

        if (schrittbegonnen) {
            if (values[2] > schwellwert) {
                schrittbegonnen = false;
                SchrittAnzahl = SchrittAnzahl + GehHeimFaktor;
                AktuelleRichtung= -AktuelleLage;

//          ... Verarbeitung ...

            }
        } else {
            if (values[2] < -schwellwert) {
                schrittbegonnen = true;
            }
        }
```

Abbildung 31: Syntax Schrittzählung

Entsprechend der Anforderung an die App soll bei einer erkannten Drehung eine Rückführung ausgelöst werden und zwischen den Modi „*** Away ***“ und „*** To go ***“ umgeschaltet werden. Dabei werden einige Parameter gesetzt, die die Verarbeitung bei der jeweiligen Gehrichtung unterstützen.

Die Drehung wurde, wie in Abbildung 32: Syntax Gehrichtung umschalten ersichtlich, mittels Cosinus Funktion realisiert. Es wird die Winkeldifferenz in Radianten verglichen, was die Abfrage vereinfacht, da mit einem konstanten Wert, in dem Fall -0,2, abgefragt wird. Math.cos erfordert die Einheit Radianten. Dies wurde hier und in weiterer Folge durch eine Umrechnung in der dargestellten Form realisiert, mal Pi durch 180.

```
//          Navigation durch Richtung und Schrittzählung: Das Umschalten zwischen de
//          Ermittlung der Drehung vom letzten Schritt zum aktuellen. Wird ein Grenz
//          den Modi getoggelt.

              if (Math.cos(((3.14159265*(AlteLage-AktuelleRichtung))/180)) < -0.2)
                  if (GehHeim) {
                      GehHeim=false;
                      GehHeimDrehung=0;
                      GehHeimFaktor=1;
                      if (offSet==false) {
                          infotextr = "***     Away     ***";
                          infotext.setText(infotextr);
                      }
                  } else
                  {
                      GehHeim=true;
                      GehHeimDrehung=180;
                      GehHeimFaktor=-1;
                      if (offSet==false) {
                          infotextr = "***     to Go     ***";
                          infotext.setText(infotextr);
                      }
                  }
              }
```

Abbildung 32: Syntax Gehrichtung umschalten

Zur Erprobung, ob die gewählte Strategie geeignet ist, wurden stichprobenartige Berechnung per Hand (und Taschenrechner) durchgeführt. Das Ergebnis war positiv. Siehe dazu Tabelle 13: Probe Richtungsänderung

Tabelle 13: Probe Richtungsänderung

AlteLage, Beispielhafte Werte	Aktuelle Lage, Beispielhafte Werte	Cosinus aus Differenz
-30	90	*-0,5 (toggle)*
0	-170	*-0,9848 (toggle)*
40	-20	*0,5 (kein toggle)*

-160	178	*0,9272 (kein toggle)*

In Abbildung 33: Syntax Bewegung aufzeichnen werden die Schritte je nach Gehrichtung zu Bewegungen nach vorne und zur Seite kumuliert. Dazu wird die aktuelle Gehrichtung herangezogen, AktuelleRichtung und auf die x- und y-Achsen des Koordinatensystems projiziert. Mit der Subtraktion um 180 zugleich die Richtung zum Ausgangspunkt errechnet.

```
//              Navigation durch Richtung und Schrittzählung: Für das Erkennen der Ri
//              genutzt und die sich durch einen Schritt daraus ergebende Projektion
//              (...zurSeite) ermittelt..

                BewegungNachVor= BewegungNachVor + Math.cos(((AktuelleRichtung-180)*
                BewegungZurSeite= BewegungZurSeite + Math.sin(((AktuelleRichtung-180)

                AlteLage=AktuelleRichtung;
```

Abbildung 33: Syntax Bewegung aufzeichnen

Auch hier wurden Tests durchgeführt, um den gewählten Rechenweg zu untermauern.

Tabelle 14: Probe Projektion auf Koordinatensystem

Aktuelle Lage, Beispielhafte Werte	Cosinus, X Achse, nachVor	Sinus, Y Achse, zurSeite
47	*0,6820*	*0,7313*
120	*-0,5*	*0,8660*
-165	*-0,9660*	*-0,2588*
-78	*0,2079*	*-0,9781*

Um aus den kumulierten Projektionen auf x- und y-Achse wiederum einen Richtungswinkel, den kumulierten Gehwinkel, zu gewinnen, wird eine Tangensfunktion genutzt, siehe Abbildung 34: Syntax Gesamtrichtung und -distanz. Zur richtigen Interpretation in den Quadranten, muss eine Korrektur durchgeführt werden.

```
//              Navigation durch Richtung und Schrittzählung: Zur Ermittlung Fortbew
//              beiden Katheten genutzt.

                if (BewegungNachVor != 0) {
                    NeueRichtung=Math.toDegrees(Math.atan(BewegungZurSeite/BewegungN

                    if (BewegungNachVor < 0) {
                        if (BewegungZurSeite<0) NeueRichtung=-180 + NeueRichtung;
                        if (BewegungZurSeite>=0) NeueRichtung=180 + NeueRichtung;
                    }
                }
//              Navigation durch Richtung und Schrittzählung: Zur Ermittlung relativ
//              wird ein Tangens aus dem Verhältnis der beiden Katheten genutzt.

                RelativeSchrittAnzahl = (float) Math.sqrt(BewegungNachVor*BewegungNa
                AktuelleLageSumme=0;
```

Abbildung 34: Syntax Gesamtrichtung und -distanz

Auch hier wurde eine Plausibilisierung vorgenommen, siehe Tabelle 15: Probe Gesamtrichtung.

Bewegung NachVor, Beispielhafte Werte	Bewegung ZurSeite, Beispielhafte Werte	NeueRichtung
25	3	*6,8427*
-6	8	*-53,1301*
70	-8	*-6,5198*
-7	-2	*15,9454*

Tabelle 15: Probe Gesamtrichtung

Zuvor wurde bereits der Schrittzähler je nach Gehrichtung hoch oder hinunter gezählt. Dies gibt zwar die gegangene Distanz wieder, muss aber nicht gleichbedeutend die Distanz zum Ausgangspunkt wiedergeben. Es könnte zum Beispiel ein Kreis gegangen worden sein. In diesem Fall reduziert sich der Abstand zum Ausgangspunkt auf die relative Schrittanzahl. Zur Berechnung wird die Hypotenuse aus den beiden Katheten ermittelt. Dazu wird wie in Abbildung 34: Syntax Gesamtrichtung und -distanz dargestellt, der pythagoreische Lehrsatz herangezogen.

4.3.4 Implementierung: Navigation durch Beschleunigung

Die Messung der Beschleunigung für den angeführten Zweck ist, wie schon zuvor ausgeführt, problematisch. Theoretisch sollte die wirkende Beschleunigung auf das Gerät eine Richtung erkennen lassen. Im praktischen Einsatz konnten jedoch keine brauchbaren Ergebnisse erzielt werden. Dies kann entweder an einem fundamentalen Fehler bei der Theorie oder auch an einem Fehler bei der Implementierung liegen.

Da es sich bei der Arbeit um eine mögliche Unterlage für den Wissensaufbau in diesem Wissensgebiet handelt, wurden die Passagen im Sourcecode lediglich auskommentiert und nicht restlos entfernt. Alle betreffenden Passagen sind mit V3 kommentiert. Siehe Abbildung 35: Syntax Richtung durch Beschleunigung. Werden diese Kommentare entfernt, ist damit die Funktion, fehlerhaft oder nicht, freigelegt.

```
//          Navigation durch Beschleunigung: Die Beschleunigung die auf x Achse (..
//          werden 2 mal integriert und das Ergebnis kumuliert. Minus deshalb, weil
//          nach hinten auf das Gerät wirkt. Nachdem die Lage des Gerätes variiert,
//          auf die Gerätelage gemessen wird, muss eine Korrketur vorgenommen werde
//          normalisiert.
//V3Info    Es handelt sich hierbei um Berechnung für die Variante 3, die in dieser

//V3    ABewegungNachVor = (float) (ABewegungNachVor + (float) (0.5 * values[0] * (
//V3    ABewegungZurSeite = (float) (ABewegungZurSeite + (float) (0.5 * values[1] *

//V3    LetzteZeit = System.currentTimeMillis();

//          Navigation durch Beschleunigung: Ermitteln des Bewegungsvektors aus der
//          Pythagoreischem Lehrsatz die Hypothenuse errechnet.

//V3        DistanzMeter = (float) Math.sqrt(ABewegungNachVor*ABewegungNachVor + AB

//          Navigation durch Beschleunigung: Zur Ermittlung Fortbewegungsrichtung i
//          beiden Katheten genutzt.

//V3     ANeueRichtung=Math.toDegrees(Math.atan(ABewegungZurSeite/ABewegungNachVor)
//V3     if (ABewegungNachVor < 0) {
//V3        if (ABewegungZurSeite<0) ANeueRichtung=-180 - ANeueRichtung;
//V3        if (ABewegungZurSeite>=0) ANeueRichtung=180 - ANeueRichtung;
//V3    }

//          Anzeige: Die Richtungsanzeige soll immer relativ zur Lage des Gerätes e

//V3    AAnzeigeRichtung=ANeueRichtung + LageZuBeginn + KompassLage + GehHeimDrehur
//V3    if (AAnzeigeRichtung<-180) AAnzeigeRichtung += 360;
//V3    if (AAnzeigeRichtung>180) AAnzeigeRichtung -= 360;
```

Abbildung 35: Syntax Richtung durch Beschleunigung

4.3.5 Implementierung: Filter

Der Beschleunigungssensor zeigt ein starkes Rauschverhalten, was bei doppelter Integration zu einer zusätzlichen Fehlerquelle führt. Um diese zu reduzieren, wurde ein gleitendes Durchschnittsverfahren zur Glättung der Messergebnisse herangezogen. Anmerkung: Auch wenn dies in der aktuellen Implementierung keine Relevanz hat, so soll es der Vollständigkeit halber erläutert werden.

Die Grundlage dafür stammt von Greg Milette /8/, Seite 107 und wurde wie in Abbildung 37: Syntax Moving Average Filter implementiert. Es handelt sich dabei um einen in seiner Länge definierbaren Puffer, aus dessen Elemente ein Durchschnittswert ermittelt wird. Durch die Veränderung der Pufferlänge kann die Filterfunktion gesteuert werden. Zur Anwendung wird wie in Abbildung 36: Syntax Aufruf Filter ein aktueller Sensorwert in den Puffer geschrieben und ein Durchschnittswert geholt. Der Einsatz muss jedoch bezogen auf den Anwendungsfall

justiert werden. So hat etwa der Filter bei Erkennung von Schritten zu Problemen geführt und wurde deshalb dafür deaktiviert.

```
//          Filterung: Um das Rauschverhalten zu minimieren wird SMA (Simple Moving
//          ein gleitendes Durchschnittsverfahren, genutzt

        softValues1.pushValue(values[0]);
        values[0] = softValues1.getValue();
        softValues2.pushValue(values[1]);
        values[1] = softValues2.getValue();

//          Filterung: Um Schritte erkennen zu können, muss SMA weg gelassen werden
//      softValues3.pushValue(values[2]);
//      values[2] = softValues3.getValue();
```

Abbildung 36: Syntax Aufruf Filter

```
public class MovingAverage {

        private float circularBuffer[] = new float [2];
        private float avg = 0;
        private int circularIndex = 0;
        private int count = 0;


        public float getValue()
        {
            return avg;
        }

        public void pushValue(float x)
        {
            if (count++ ==0)
            {
                primeBuffer(x);
            }
            float lastValue=circularBuffer[circularIndex];
            avg = avg + (x - lastValue) / circularBuffer.length;
            circularBuffer[circularIndex] = x;
            circularIndex = nextIndex(circularIndex);
```

Abbildung 37: Syntax Moving Average Filter

4.3.6 Implementierung: Anzeige

Das Koordinatensystem ist geräteunabhängig und orientiert sich an einer Nordausrichtung. Für die Anzeige muss wiederum eine Umrechnung auf die aktuelle Gerätelage erfolgen.

Zur Ausgabe der Werte, wurde eine Routine eingerichtet, die das Update des Displays vornimmt. Auf die Zeitsteuerung wurde bereits hingewiesen. Darauf wird an dieser Stelle nicht näher eingegangen und auf den Sourcecode verwiesen.

4.3.7 Implementierung: Protokollierung

Um Messreihen protokolliert durchführen zu können, werden während der Messung die Daten in ein Array geschrieben und bei OnDestroy auf einen externen Datenträger ausgege-

ben. Mittels USB kann die csv-Datei vom Computer geöffnet und in Microsoft Excel weiterverarbeitet werden.

Abbildung 39: Syntax Protokolldaten sammeln zeigt die laufende Befüllung des Arrays, Abbildung 40: Syntax Protokolldaten wegschreiben das Wegschreiben der Werte am Ende und Abbildung 38: Syntax externe Protokollierung das Öffnen zu Beginn. Die Zwischenspeicherung in ein Array wurde aus Performancegründen realisiert, um ein rasches Wegschreiben während der Sensorereignisse zu ermöglichen.

```
//          Protokollierung: Es werden laufend Sensordaten in ein Array zwischen
//          am Ende auf einen externen Speicher fortgeschrieben. Hier wird geprü
//          Speicher zum Schreiben verfügbar ist. Wenn ja, wird eine Datei angel

        String state = Environment.getExternalStorageState();
        if (Environment.MEDIA_MOUNTED.equals(state)) {
            mExternalStorageWriteable = true;
            dir.mkdirs();
       }
```

Abbildung 38: Syntax externe Protokollierung

Es besteht hierbei relativ einfach die Möglichkeit Sensordaten zu ergänzen oder zu entfernen und damit eine indiviudelle Gestaltung der Protokollierung für die eigenen Untersuchungszwecke zu ermöglichen.

```
//              Protokollierung: Befüllt die Messreihen-Datei für anschließend weite
        StepCount++;
        csvString[StepCount] = Float.toString(StepCount)+";"+Float.toString(values[0
```

Abbildung 39: Syntax Protokolldaten sammeln

Ein Ansprechen ist über die USB Schnittstelle möglich. Die Datei wird im Verzeichnis navigate gespeichert.

```
//          Protokollierung: Die zwischengespeicherten Sensordaten werden auf den e
//          geschrieben.

        if (mExternalStorageWriteable) {
            try {
                FileOutputStream f = new FileOutputStream(file);
                PrintWriter pw = new PrintWriter(f);
                csvString[0] = ";A x-Achse;A y-Achse ; A z-Achse; ABewegungNachVor;
                for(int i=0;i<csvString.length;i++)
                    pw.println(csvString[i]);
                pw.println(csvString);
                pw.flush();
                pw.close();
                f.close();
                }
                catch (FileNotFoundException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
```

Abbildung 40: Syntax Protokolldaten wegschreiben

4.4 Praktische Erfahrungen und Messreihen

Für die Erprobung im praktischen Anwendungsfall wurden mehrere Testreihen durchgeführt. Dabei wurde versucht unterschiedliche Szenarien abzubilden, die vielleicht auftreten können. Ein großer gegangener Kreis (Kreisgang) soll die relative Schrittanzahl auf null bringen. Es wird auch der eigentliche Zweck der Applikation, die Rückführung zu einem Ausgangspunkt, getestet. Hierbei wird in eine einfache und komplexe Rückführungsaufgabe unterschieden. Die Messergebnisse und die Anmerkungen zu den vorgenommenen Tests werden ebenfalls hier angeführt.

Die Tests fanden in 1130 Wien, Österreich, statt. Am 27. September 2012, in der Zeit von 19:30 bis 20:30. Die Abbildung 41: Testgelände zeigt das Areal, wo die Tests stattgefunden haben. Es wurden 3 unterschiedliche Parcours bezwungen. Der Kreisgang, die einfache Rückführung (rote Markierung) und die komplexe Rückführung.

Die Erklärung der Tests und die Ergebnisse sind in diesem Kapitel erläutert.

Abbildung 41: Testgelände

4.4.1 Test: Kreisgang

Beim Kreisgang, eigentlich ein Dreieck, wird ausgehend von einem Ausgangspunkt eine Strecke gegangen und wieder zum Ausgangspunkt zurück, ohne dass der gleiche Weg genommen wird. Die Schrittanzahl wird beim Erreichen abgelesen.

Test	Schrittanzahl	Schrittabweichung (gerundet)	Dauer (Sekunden)	Beurteilung	Anmerkung
KG1	107	24	00:59	Genügend	Kalibrierung nicht ganz abgewartet
KG2	109	21	01:05	Befriedigend	
KG3	103	18,5	01:15	Gut	Beim Start bereits auf To go geschaltet
KG4	110	23	01:02	Befriedigend	Dto

KG5	115	4	01:03	Sehr gut	Dto.
KG6	110	4	1:05	Sehr gut	

Tabelle 16: Messergebnis Kreisgang

Das Ergebnis kann insgesamt als erfolgreich angesehen werden. Es stellt zumindest dar, dass die Funktionsweise prinzipiell gegeben ist. Man kann erkennen, dass zwischen den Ergebnissen KG1 bis KG4 und KG5/KG6 erhebliche Unterschiede sind. Dies liegt an der Gehrichtung. Die ersten Runden wurden im Uhrzeigersinn gegangen, während die anderen gegen den Uhrzeigersinn gegangen wurden. Der Drift des Gyroskops, der auch in Ruhelage auffällt, kommt der einen Gehrichtung offensichtlich entgegen und schadet der anderen mehr.

Weiters ist aufgefallen, dass 3 mal die Umschaltung auf „To go“ aktiviert wurde, was eigentlich eine Fehlfunktion ist oder durch einen falschen Initialwert ausgelöst wird.

4.4.2 Test: Einfache Rückführung

Bei der einfachen Rückführung wurde eine annähernd gerade Wegstrecke zurück gelegt, rechts abgebogen, etwa nochmals die gleiche Wegstrecke gegangen und umgekehrt. Es wurde dann auf dem Weg entlang gegangen und sobald es möglich war, der Richtungsnadel gefolgt. Solange bis die Anzeige annähernd 0 angezeigt hat. Ganz 0 wurde nie angezeigt. Die zurück gelegte Wegstrecke in Schritten lag nicht vor, weil ab der Umkehrung in den „To go“ Modus herunter gezählt wurde und somit im Zielbereich keine valide Schrittanzahl angezeigt wurde. Die Schrittanzahl zum tatsächlichen Ausgangspunkt wurde abgegangen und die Schritte gezählt.

Test	Schrittanzahl	Schrittabweichung (gerundet)	Dauer (Sekunden)	Beurteilung	Anmerkung
ER1	Unbekannt	5	00:51	Sehr gut	
ER2	Unbekannt	9	00:56	Befriedigend	Keine „To go“ Umschaltung
ER3	Unbekannt	6	00:55	Sehr gut	

Tabelle 17: Messergebnis einfach Rückführung

Der Test war erfolgreich. Die Umschaltung einmal nicht funktioniert. In allen Fällen konnte man sehr nah den Ausgangspunkt herankommen. Beim Rückweg wurde erst relativ spät auf die Richtungsnadel geachtet, nachdem der Weg vorgegeben war. Hier muss angemerkt werden, dass das Gehen nach der Richtungsnadel sehr schwierig ist, sehr volatil und scheinbar sehr inkonstant. Zeitweise scheint die Richtungsangabe „verwirrt“. Generell muss darauf geachtete werden, dass nicht versucht wird, die Richtung des Smartphones ohne

Richtungsänderung des gesamten Körpers zu ändern, auch wenn man bemüht ist die vorgegebene Richtung zu gehen.

4.4.3 Test: Komplexe Rückführung

Bei diesem Test wurde ein Parcours mit eine Gehdauer von annähernd drei Minuten abgegangen. Im letzten Drittel wurde eine Umkehrung vorgenommen und die Rückführung mit Distanz und Richtungsnadel versucht. Die Übung wurde einmal mit dem Uhrzeigersinn und dann dagegen durchgeführt. Es wurde gemäß Rückführung gegangen, solange bis der erkannte Ausgangspunkt erreicht wurde. Die Abweichung wurde abgegangen und die Schritte gezählt.

Test	Schrittanzahl	Schrittabweichung (gerundet)	Dauer (Sekunden)	Beurteilung	Anmerkung
KR1	Unbekannt	23	02:45	Genügend	
KR2	Unbekannt	25	02:55	Genügend	Nach dem Umkehrpunkt zunächst Zeiger „verwirrt“

Tabelle 18: Messergebnis komplexe Rückführung

Die Distanzabweichung scheint für die Dauer relativ gut und brauchbar. Die Nutzung der Richtungsnadel ist aber ebenso wie bei der einfachen Rückführung schwierig. Es hat sich ein ähnliches Verhalten gezeigt. Zeitweise ist die Richtungsangabe „verwirrt“, zeigt in falsche Richtung und dann wieder in die richtige. Ohne Kenntnis des Ausgangspunktes und somit unbewusster Manipulation ist ein zuverlässiges Zurückfinden zum Ausgangspunkt nicht möglich.

4.5 Reflexion zum Anwendungsbeispiel und zum Prototypen

Das Beispiel zeigt, dass trotz einfacher Aufgabenstellung rasch ein komplexes Thema entstehen kann. Die Unabhängigkeit zwischen Lage des Geräts, Bewegungsrichtung und der Darstellung auf dem Display sind einige der Herausforderungen.

Das Auslesen der Sensoren hat sich als sehr einfach erwiesen. Einfacher als erwartet. Während die Auswertung und die Interpretation der Werte unterschätzt wurde.

Das Smartphone muss zu Beginn mit einem Referenzsystem gekoppelt werden. In diesem Fall der Kompass. Ab dann werden alle Bewegungen über den Drehratensensor und dem Integral der Drehgeschwindigkeit autonom ermittelt. Die Ergebnisse werden auf ein Koordinatensystem aufgetragen, das sich an Norden orientiert, egal in welcher Lage sich das Smartphone befindet. Die Bewegungen werden kumuliert und der Ausgangspunkt angezeigt.

Dies aber wiederum abhängig von der Lage des Smartphones. Durch die 3 übereinander gelagerten Ebenen(Smartphone, Koordinatensystem, Anzeige) entsteht eine Komplexität, die schnell zu Fehlern führen kann.

Zur Berechnung zwischen Fortbewegung im Koordinatensystem und den Winkelabweichungen wurden Winkelfunktionen herangezogen. Auch hier sind Fehlerquellen verborgen. Insbesondere weil nach 360 wieder 0 beginnt nach minus 180 179. All das muss bei Rechenoperationen berücksichtigt werden.

Der Drift oder Bias den das Gyroskop aufweist führt zu Abweichungen und im Laufe der Zeit zu erheblicher Ungenauigkeit. Eine entsprechende Korrektur ist hier erforderlich. Generell müsste für eine sinnvolle Anwendbarkeit der Prototyp überarbeitet werden, um höhere Genauigkeit zu erzielen. Das Richtungsanzeigethema müsste ebenfalls adressiert werden.

In der Handhabung ist darauf zu achten, dass das Smartphone in einer stabilen, konstanten Lage gegenüber dem menschlichen Körper bleibt. Gerade bei der Rückführung ist man geneigt, die Richtungsbewegungen nur mit dem Gerät, nicht aber mit dem gesamten Körper zu vollziehen. Auch ist zu beachten, dass die Schrittrichtung abgeleitet wird, in dem Moment, wo ein Schritt als abgeschlossen erkannt wird.

Die Umschaltung zwischen „Away“ und „To go“ müsste ebenfalls nochmals untersucht werden, ob hier Schwellwerte optimiert werden müssen. Hier wurde mehrfach eine Fehlfunktion festgestellt.

Neben den qualitätsverbessernden Maßnahmen sind funktionale und ergonomische Erweiterungen angebracht. Die Aufnahme des Barometers für die Abschätzung der Höhendifferenz erscheint nützlich. Im vorliegenden Gerät war jedoch kein Sensor eingebaut. Aus der Dokumentation von Android kann eine einfache Handhabung abgeleitet werden. Eine kurzzeitige Luftdruckänderung könnte somit als Höhenbewegung interpretiert werden.

Die Darstellung auf dem Display müsste angepasst werden, sodass ein einfaches Ablesen möglich ist. Die Sensor-Rohdaten können weggelassen werden. Die Zielerreichung sollte durch ein akustisches oder visuelles Signal leicht erkennbar sein. In den Tests wurde nie genau 0 getroffen, aber das Durchgehen durch den Nullpunkt wurde mit „Zeigerrotation“ sichtbar.

5 Bewertung und Ausblick

Die aktuelle Generation von Android Smartphones, eine einfach zugängliche und vollkommen offenen Plattform, bietet unbegrenzte Möglichkeiten für den kreativen Einsatz, der in den Geräten enthaltenen Komponenten.

Die erforderlichen Skills für die Java Entwicklung und das Verständnis zu den bereits von Android bereitgestellten APIs und fertigen Softwarekomponenten ist aber nicht zu unterschätzen. Gemeinsam mit den Anforderungen zur Behandlung von Navigationsaufgaben und der Begegnung von Ungenauigkeitsproblemen sind entsprechende Fähigkeiten gefordert.

Es hat sich gezeigt, dass Beschleunigungssensoren sehr problematisch hinsichtlich Aufzeichnung von Wegstrecken sind. Insbesondere dann, wenn das Messgerät von einem gehenden Menschen getragen wird. Die Erschütterungen durch Schritte überlagern alle anderen Messergebnisse. Weiters birgt die doppelte Integration ein großes Fehlerpotential. Hingegen erfolgt die Schrittzählung sehr zuverlässig. Die Richtungserkennung hätte sollen durch Beschleunigungsdaten indiziert werden sollen, was sich als untauglich heraus stellte. Die Messung der Lage mittels Kompass war zwar in manchen Situationen verwendbar, jedoch zu wenig zuverlässig und zu träge. Erst der Einsatz des Gyroskops brachte zuverlässige und verwertbare Resultate.

Ein Prototyp konnte hergestellt werden, der Tests standhielt. Wenn auch eine Praxistauglichkeit nur eingeschränkt gegeben ist, so kann es zumindest als Ausgangsbasis für alternative oder ähnliche Aufgabenstellungen dienen.

Die technischen Aspekte zur Sensorik wurden ausführlich erläutert und sollten dem Leser in kompakter Form die Funktionsweise von Beschleunigungs- und Drehratensensoren und deren Implementierungsformen in aktuellen Smartphones näher bringen.

Zusammen mit den bereitgestellten Informationen zu Android, der Art der Ansteuerung von Sensoren sowie der Erklärung der Klassen und Methoden, die für dieses Einsatzgebiet wichtig erscheinen, sollte für Studenten und Interessierte ein guter Grundstock gelegt sein, Aufgabenstellungen in diesem Bereich umzusetzen.

Mit dem Beispiel „Guide Me Home“ wurden die Problemstellungen beleuchtet, die bei einer praktischen Umsetzung auftreten können und eine Grundlage geschaffen, auf der weitere Untersuchungen möglich sind.

6 Literatur

/1/ http://developer.android.com, gesehen am 14.6.2012

/2/ Eichler; Physik: Für das Ingenieurstudium prägnant mit knapp 300 Beispielen
Gabler Wissenschaftsverlage, 2011

/3/ Henn; Gyroskope in Raketen: Werden Faseroptische Kreisel eingesetzt?
Grin Verlag, 2010

/4/ Hoffmann; Taschenbuch der Messtechnik
Hanser Verlag, 2011

/5/ Komatineni; Pro Android 4
New York, 2012

/6/ Lechner; Trägheitsnavigation
VDM Verlag, 2010

/7/ Meier; Professional Android 4 Application Development
John Wiley and Sons, 2012

/8/ Milette, Stroud; Android Sensor Programming
John Wiley and Sons, 2012

/9/ Post; Android Apps entwickeln
Galileo Press, 2012

/10/ Reif; Automobilelektronik: Einführung für Ingenieure
Springer, 2012

/11/ Schiessle; Industriesensorik
Vogel Business Media, 2010

/12/ Wendel; Integrierte Navigationssysteme, Sensordatenfusion
Oldenburg Verlag, 2011

7 Anhänge

7.1 NavigateActivity.java

```
package at.ronkdev.gmh;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintWriter;

import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.widget.TextView;
```

```
public class NavigateActivity extends Activity implements SensorEventListener, Runnable {

    private SensorManager mSensorManager;

    private RichtungView kompassnadel;
    private TextView richtungwert;
    private TextView distanzwert;
    private TextView richtungwertS;
    private TextView distanzwertS;
    private TextView reldistanzwert;
    private TextView linawert1;
    private TextView linawert2;
    private TextView linawert3;
    private TextView rotvwert1;
    private TextView rotvwert2;
    private TextView rotvwert3;
    private TextView gyrowert1;
    private TextView gyrowert2;
    private TextView gyrowert3;
    private TextView zaehlerwert1;
    private TextView zaehlerwert2;
    private TextView magfwert1;
    private TextView magfwert2;
    private TextView magfwert3;
    private TextView infotext;

    private float[] linawert = new float[3];
    private int StepCount = 0;
```

```
    private float[] AusgleichsLage = new float[3];
    private float[] RotationsMatrix = new float[16];
       private int zaehl = 0;
       private static final float KonstanteSchwellwert = 1.0f;
       private float schwellwert = KonstanteSchwellwert;
       private boolean schrittbegonnen = false;
       private double LetzteZeit, GLetzteZeit;


       private double DistanzMeter=0;
       private int GehHeimDrehung=180;
       private int GehHeimFaktor=1;
       private double KompassLage, LageZuBeginn, AktuelleLage, AlteLage, NeueRichtung, ANeueRichtung;
       private double BewegungNachVor, BewegungZurSeite, AnzeigeRichtung, AAnzeigeRichtung, ABewegungNachVor, ABewegungZurSeite,
GBewegungZurSeite;
       private double AktuelleLageSumme, AktuelleRichtung;
       private float magfwertv1=0;


       private float magfwertv2=0;
       private float magfwertv3=0;
       private float rotvwertv1=0;
       private float rotvwertv2=0;
       private float rotvwertv3=0;
       private float gyrowertv1=0;
       private float gyrowertv2=0;
       private float gyrowertv3=0;
       private float RelativeSchrittAnzahl=0, SchrittAnzahl=0;
       private float OffSet1=0, OffSet2=0, OffSet3=0;
       private float OffSet1Summe=0, OffSet2Summe=0, OffSet3Summe=0;
```

```
   private static WakeLock myWakeLock;
private boolean refreshing=true, GehHeim=false;
private boolean offSet=true;
private boolean mExternalStorageWriteable = false;
   String[] csvString = new String[20000];
private String infotextr="***     Away     ***";

File root = android.os.Environment.getExternalStorageDirectory();

File dir = new File (root.getAbsolutePath() + "/navigate");
File file = new File(dir, "navigate_versuchsreihe.csv");

MovingAverage softValues1 = new MovingAverage();
MovingAverage softValues2 = new MovingAverage();
MovingAverage softValues3 = new MovingAverage();

private double startzeit = System.currentTimeMillis();

@Override
protected void onCreate(Bundle b) {
    super.onCreate(b);

//                  Protokollierung: Es werden laufend Sensordaten in ein Array zwischengespeichert und
//                  am Ende auf einen externen Speicher fortgeschrieben. Hier wird geprüft, ob der externe
//                  Speicher zum Schreiben verfügbar ist. Wenn ja, wird eine Datei angelegt.

    String state = Environment.getExternalStorageState();
    if (Environment.MEDIA_MOUNTED.equals(state)) {
```

```
            mExternalStorageWriteable = true;
             dir.mkdirs();
       }

//                  Anzeige: Es werden die Felder initialisiert, die laufend mit Werten
//           befüllt werden und periodisch angezeigt werden.

        setContentView(R.layout.main);

     richtungwert = (TextView) this.findViewById(R.id.richtungwert);
     distanzwert= (TextView) this.findViewById(R.id.distanzwert);
     reldistanzwert= (TextView) this.findViewById(R.id.reldistanzwert);
     richtungwertS = (TextView) this.findViewById(R.id.richtungwertS);
     distanzwertS= (TextView) this.findViewById(R.id.distanzwertS);
     linawert1 = (TextView) this.findViewById(R.id.linawert1);
     linawert2 = (TextView) this.findViewById(R.id.linawert2);
     linawert3 = (TextView) this.findViewById(R.id.linawert3);
     rotvwert1 = (TextView) this.findViewById(R.id.rotvwert1);
     rotvwert2 = (TextView) this.findViewById(R.id.rotvwert2);
     rotvwert3 = (TextView) this.findViewById(R.id.rotvwert3);
     gyrowert1 = (TextView) this.findViewById(R.id.gyrowert1);
     gyrowert2 = (TextView) this.findViewById(R.id.gyrowert2);
     gyrowert3 = (TextView) this.findViewById(R.id.gyrowert3);
     zaehlerwert1 = (TextView) this.findViewById(R.id.zaehlerwert1);
     zaehlerwert2 = (TextView) this.findViewById(R.id.zaehlerwert2);
     magfwert1 = (TextView) this.findViewById(R.id.magfwert1);
     magfwert2 = (TextView) this.findViewById(R.id.magfwert2);
     magfwert3 = (TextView) this.findViewById(R.id.magfwert3);
```

```
      infotext = (TextView) this.findViewById(R.id.infotext);
      kompassnadel = (RichtungView) this.findViewById(R.id.compassview);


//                Sensoren: Der Sensormanager wird aktiviert

      mSensorManager = (SensorManager) this.getSystemService(Context.SENSOR_SERVICE);

//                  Anzeige: Damit der Bildschirm während der Verarbeitung nicht gesperrt wird, wird
//                  diese Funktion gesperrt.

       if (myWakeLock==null) {
                  final PowerManager pm = (PowerManager) (getSystemService(Context.POWER_SERVICE));
                  myWakeLock = pm.newWakeLock(PowerManager.SCREEN_BRIGHT_WAKE_LOCK,
                              getString(R.string.app_name));
             }
             myWakeLock.acquire();

//                  Anzeige: Um nicht bei jedem Sensorevent eine Bildschirmausgabe zu erzwingen, erfolgt
//                  diese durch einen separaten Thread, der zeitgesteuert aufgerufen wird.

             Thread thread = new Thread(this);
             thread.setName("GUIRefreshThread");
             thread.start();

    }

       @Override
```

```
	protected void onResume() {
		super.onResume();

//                  Sensoren: Die erforderlichen, gewünschten Sensoren werden für die Verwendung registriert.

		mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION), SensorMana-
ger.SENSOR_DELAY_FASTEST);
		mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR), SensorMana-
ger.SENSOR_DELAY_FASTEST);
		mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD), SensorMana-
ger.SENSOR_DELAY_UI);
		mSensorManager.registerListener(this, mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE), SensorMana-
ger.SENSOR_DELAY_UI);
		refreshing=true;

	}

	@Override
	protected void onPause() {
//                  Sensoren: Werden hier bewusst nicht un-registriert, da sonst die aufgezeichnete Distanz
//                  verloren gehen würde. Es wird lediglich die Anzeige unterbunden.

	  refreshing=false;
		super.onPause();
	}

	@Override
	protected void onDestroy() {
//                  Sensoren: Erst hier werden die Sensoren un-registriert...
```

```
            mSensorManager.unregisterListener(this);
          refreshing=false;
          super.onPause();
            if(myWakeLock!=null) {
            myWakeLock.release();
        }
//                  Protokollierung: Die zwischengespeicherten Sensordaten werden auf den externen Datenspeicher
//            geschrieben.

            if (mExternalStorageWriteable) {
           try {
               FileOutputStream f = new FileOutputStream(file);
               PrintWriter pw = new PrintWriter(f);
               csvString[0] = ";A x-Achse;A y-Achse ; A z-Achse; ABewegungNachVor; ABewegungZurSeite; DistanzMeter; Kompass; Gyro
x-Achse; Gyro y-Achse; Gyro z-Achse";
                   for(int i=0;i<csvString.length;i++)
                         pw.println(csvString[i]);
               pw.println(csvString);
               pw.flush();
               pw.close();
               f.close();
            }
            catch (FileNotFoundException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                              e.printStackTrace();
                        }
```

```
		}

	}

	public void onSensorChanged(SensorEvent event) {

//			Sensoren: Das Kernstück. Sobald ein Sensoren-Ereignis auftritt, wird diese Routine
//			angesprochen. Darin wird abgefragt, welcher Sensor das Ereignis ausgelöst hat und die
//			entsprechenden Verarbeitungsschritte durchgeführt.

//			Für die Navigationsaufgaben werden der Lineare Beschleunigungssensor, Gyroskop und der
//			Rotationsvektorsensor genutzt. Die anderen liefern lediglich Sensor-Rohdaten
//			zur Information.

		if (LetzteZeit==0) LetzteZeit = System.currentTimeMillis();

	switch (event.sensor.getType()) {
		case Sensor.TYPE_LINEAR_ACCELERATION:
				do_acc_calc(event.values);
			break;
		case Sensor.TYPE_ROTATION_VECTOR:
				do_rotvect_calc(event.values);
			break;
		case Sensor.TYPE_MAGNETIC_FIELD:
				magfwertv1=event.values[0];
				magfwertv2=event.values[1];
				magfwertv3=event.values[2];
			break;
```

```
            case Sensor.TYPE_GYROSCOPE:

//                          Sensoren: Mit dem Gyroskop wird die Drehrate des Gerätes gemessen. Das Ergebnis
//                          sind Radianten je Zeit. Durch Integration kann der zurück gelegte Winkel ermittel
//                          und damit die Lageänderung.
//                          Zur einfachen Ermittlung des Winkels wurde durch einen Versuch eine Konstante ermittelt.
//                          Das Ergebnis wird in AktuelleLage abgelegt.

                            gyrowertv1=event.values[0];
                            gyrowertv2=event.values[1];
                            gyrowertv3=event.values[2];
                            if (offSet) {
                                  AlteLage=AktuelleLage;
                                  GLetzteZeit = System.currentTimeMillis();
                            }
                            GBewegungZurSeite = (float) (GBewegungZurSeite + (float) (event.values[2] * ((System.currentTimeMillis()-
GLetzteZeit)/1000)));
                            AktuelleLage = (-LageZuBeginn-(GBewegungZurSeite/(-0.015896989)))%360;
                            GLetzteZeit=System.currentTimeMillis();
            break;
      }
   }

   private void do_rotvect_calc(float[] values) {

//                  Sensoren: Mit dem Rotationsvektorsensor kann die Lage des Gerätes festegestellt
//                  werden. Verwertbar ist hierbei der Azimuth-Wert aus getOrientation (Wert 0), der
//                  der für die Kompassfunktion genutzt wird.
```

```
//                  Der Rotationsvektorsensor ist ein virtueller Sensor, der aus anderen, physischen
//                  Sensoren ermittelt wird (Magnetischer Feldsensor, Gyroskope und Beschleunigungssensor)
//V2Info      Der Rotationsvektorsensor wird primär für die einmalige Kalibrierung mit dem
//V2Info      magnetischen Nordpol verwendet. In der Variante V2 wird auch die Lageänderung
//V2Info      damit ausgewertet.

       rotvwertv1=values[0];
       rotvwertv2=values[1];
       rotvwertv3=values[2];

           mSensorManager.getRotationMatrixFromVector(RotationsMatrix, values);
              mSensorManager.getOrientation(RotationsMatrix, AusgleichsLage);
              KompassLage = Math.toDegrees(-AusgleichsLage[0]);
//V2   AktuelleLage = KompassLage
              if (offSet) {
                    LageZuBeginn = -KompassLage;
              }
       }

    private void do_acc_calc(float[] values) {

//                  Filterung: Um das Rauschverhalten zu minimieren wird SMA (Simple Moving Average),
//            ein gleitendes Durchschnittsverfahren, genutzt

       softValues1.pushValue(values[0]);
              values[0] = softValues1.getValue();
       softValues2.pushValue(values[1]);
              values[1] = softValues2.getValue();
```

```
//                  Filterung: Um Schritte erkennen zu können, muss SMA weg gelassen werden
//     softValues3.pushValue(values[2]);
//            values[2] = softValues3.getValue();

//                  Sensoren: Während der ersten x Millisekunden wird ein Offset berechnet, daher sollte das Smartphone
//                  ruhig und flach liegen.

            if (offSet) {
                  if ((startzeit + 5000) > System.currentTimeMillis()) {
                        zaehl=zaehl+1;
                        OffSet1Summe = (OffSet1Summe + values[0]);
                        OffSet2Summe = (OffSet2Summe + values[1]);
                        OffSet3Summe = (OffSet3Summe + values[2]);
                        } else
                        {
                        offSet = false;
                        OffSet1 = OffSet1Summe / zaehl;
                        OffSet2 = OffSet2Summe / zaehl;
                        OffSet3 = OffSet3Summe / zaehl;
                        }
            }

            values[0] = values[0]-OffSet1;
            values[1] = values[1]-OffSet2;
            values[2] = values[2]-OffSet3;

            linawert[0] = values[0];
```

```
            linawert[1] = values[1];
            linawert[2] = values[2];

//                  Navigation durch Beschleunigung: Die Beschleunigung die auf x Achse (...NachVor) und y Achse (...ZurSeite)
//                  werden 2 mal integriert und das Ergebnis kumuliert. Minus deshalb, weil bei Bewegung nach vorne eine Beschleuni-
gung
//                  nach hinten auf das Gerät wirkt. Nachdem die Lage des Gerätes variiert, jedoch die Beschleunigung immer bezogen
auf
//                  auf die Gerätelage gemessen wird, muss eine Korrketur vorgenommen werden. Es wird dazu auf die Lage zu Beginn
//                  normalisiert.
//V3Info      Es handelt sich hierbei um Berechnung für die Variante 3, die in dieser Fassung nicht berechnet wird.


//V3   ABewegungNachVor = (float) (ABewegungNachVor + (float) (0.5 * values[0] * ((System.currentTimeMillis()-
LetzteZeit)/1000)*((System.currentTimeMillis()-LetzteZeit)/1000)));
//V3   ABewegungZurSeite = (float) (ABewegungZurSeite + (float) (0.5 * values[1] * ((System.currentTimeMillis()-
LetzteZeit)/1000)*((System.currentTimeMillis()-LetzteZeit)/1000)));


//V3   LetzteZeit = System.currentTimeMillis();


//                  Navigation durch Beschleunigung: Ermitteln des Bewegungsvektors aus den kumulierten Bewegungen. Dazu wird mit-
tels
//                  Pythagoreischem Lehrsatz die Hypothenuse errechnet.


//V3       DistanzMeter = (float) Math.sqrt(ABewegungNachVor*ABewegungNachVor + ABewegungZurSeite*ABewegungZurSeite);


//                  Navigation durch Beschleunigung: Zur Ermittlung Fortbewegungsrichtung in Grad, wird ein Tangens aus dem Verhält-
nis der
//                  beiden Katheten genutzt.


//V3     ANeueRichtung=Math.toDegrees(Math.atan(ABewegungZurSeite/ABewegungNachVor));
```

```
//V3     if (ABewegungNachVor < 0) {
//V3         if (ABewegungZurSeite<0) ANeueRichtung=-180 - ANeueRichtung;
//V3         if (ABewegungZurSeite>=0) ANeueRichtung=180 - ANeueRichtung;
//V3    }

//                  Anzeige: Die Richtungsanzeige soll immer relativ zur Lage des Gerätes erfolgen

//V3    AAnzeigeRichtung=ANeueRichtung + LageZuBeginn + KompassLage + GehHeimDrehung;
//V3    if (AAnzeigeRichtung<-180) AAnzeigeRichtung += 360;
//V3   if (AAnzeigeRichtung>180) AAnzeigeRichtung -= 360;


//                  Navigation durch Richtung und Schrittzählung: Es wird die senkrechte Beschleunigung, die bei einem Schritt
//            auftritt, gemessen. Wird ein Schwellwert unterschritten wird ein Schritt begonnen, wird ein Schwellwert überschritten,
//            wird der Schritt beendet und alle notwendigen Berechnungen durchgeführt.

           if (schrittbegonnen) {
              if (values[2] > schwellwert) {
                     schrittbegonnen = false;
                     SchrittAnzahl = SchrittAnzahl + GehHeimFaktor;
                     AktuelleRichtung= -AktuelleLage;


//                  Navigation durch Richtung und Schrittzählung: Das Umschalten zwischen den Modi "Away" und "To go" erfolgt durch
//                  Ermittlung der Drehung vom letzten Schritt zum aktuellen. Wird ein Grenzwert überschritten, so wird zwischen
//                  den Modi getoggelt.

                     if (Math.cos(((3.14159265*(AlteLage-AktuelleRichtung))/180)) < -0.2) {
```

```
                if (GehHeim) {
                        GehHeim=false;
                        GehHeimDrehung=0;
                        GehHeimFaktor=1;
                        if (offSet==false) {
                                infotextr = "***     Away     ***";
                                infotext.setText(infotextr);
                        }
                } else
                {
                        GehHeim=true;
                        GehHeimDrehung=180;
                        GehHeimFaktor=-1;
                        if (offSet==false) {
                                infotextr = "***     to Go     ***";
                                infotext.setText(infotextr);
                        }
                }
        }

//                      Navigation durch Richtung und Schrittzählung: Für das Erkennen der Richtung wird der Kompass (getOrienta-
tion)
//              genutzt und die sich durch einen Schritt daraus ergebende Projektion auf die x Achse (..nachVor) und y Achse
//              (...zurSeite) ermittelt..

        BewegungNachVor= BewegungNachVor + Math.cos(((AktuelleRichtung-180)*3.14159265)/180);
      BewegungZurSeite= BewegungZurSeite + Math.sin(((AktuelleRichtung-180)*3.14159265)/180);
```

```
                AlteLage=AktuelleRichtung;

//                  Navigation durch Richtung und Schrittzählung: Zur Ermittlung Fortbewegungsrichtung in Grad, wird ein Tangens aus dem Verhältnis der
//                        beiden Katheten genutzt.

                if (BewegungNachVor != 0) {
                   NeueRichtung=Math.toDegrees(Math.atan(BewegungZurSeite/BewegungNachVor));

                   if (BewegungNachVor < 0) {
                         if (BewegungZurSeite<0) NeueRichtung=-180 + NeueRichtung;
                         if (BewegungZurSeite>=0) NeueRichtung=180 + NeueRichtung;
                   }
                }
//                  Navigation durch Richtung und Schrittzählung: Zur Ermittlung relativen Schrittanzahl, der kürzeste Weg zum Ausgangspunkt,
//                wird ein Tangens aus dem Verhältnis der beiden Katheten genutzt.

                RelativeSchrittAnzahl = (float) Math.sqrt(BewegungNachVor*BewegungNachVor + BewegungZurSeite*BewegungZurSeite);
                AktuelleLageSumme=0;

             } else {
                   AktuelleLageSumme = AktuelleLageSumme + AktuelleLage;
             }
          } else {
             if (values[2] < -schwellwert) {
                   schrittbegonnen = true;
             }
          }
```

```
//                          Protokollierung: Befüllt die Messreihen-Datei für anschließend weitere Auswertungen
             StepCount++;
        csvString[StepCount] = Float.toString(StepCount)+";"+Float.toString(values[0])+";"+
Float.toString(values[1])+";"+Float.toString(values[2])+";"+Double.toString(ABewegungNachVor)+";"+Double.toString(ABewegungZurSeite)
+";"+Double.toString(DistanzMeter)+";"+Double.toString(KompassLage)+";"+Float.toString(gyrowertv1)+";"+Float.toString(gyrowertv2)+";
"+Float.toString(gyrowertv3)+";"+Double.toString(GBewegungZurSeite);


        //          Anzeige: Die Richtungsanzeige soll immer relativ zur Lage des Gerätes und absolut zum Koordinatensystem erfol-
gen, auch dann wenn kein Schritt vorkommt


           AnzeigeRichtung=NeueRichtung + AktuelleLage;


           if (AnzeigeRichtung<-180) AnzeigeRichtung += 360;
           if (AnzeigeRichtung>180) AnzeigeRichtung -= 360;


    }


    public void updateGui () {


       richtungwert.setText(Double.toString(NeueRichtung));
       distanzwert.setText(Float.toString(SchrittAnzahl));
       reldistanzwert.setText(Float.toString(RelativeSchrittAnzahl*GehHeimFaktor));


 //           richtungwertS.setText(Double.toString(NeueRichtung));
 //           distanzwertS.setText(Double.toString(DistanzMeter));


       linawert1.setText(Float.toString(linawert[0]));
       linawert2.setText(Float.toString(linawert[1]));
```

```
      linawert3.setText(Float.toString(linawert[2]));

      rotvwert1.setText(Float.toString(rotvwertv1));
       rotvwert2.setText(Float.toString(rotvwertv2));
       rotvwert3.setText(Float.toString(rotvwertv3));

       gyrowert1.setText(Float.toString(gyrowertv1));
      gyrowert2.setText(Float.toString(gyrowertv2));
       gyrowert3.setText(Float.toString(gyrowertv3));

      zaehlerwert1.setText(Double.toString(LetzteZeit));
       zaehlerwert2.setText(Float.toString(StepCount));

            magfwert1.setText(Float.toString(magfwertv1));
            magfwert2.setText(Float.toString(magfwertv2));
      magfwert3.setText(Float.toString(magfwertv3));

      if (offSet == false) infotext.setText(infotextr);

      kompassnadel.setWinkel1( (float) KompassLage);             // weiß
//          kompassnadel.setWinkel2( (float) (NeueRichtung));    // gelb
      kompassnadel.setWinkel3( (float) AnzeigeRichtung);   // grün

   }

      public void onAccuracyChanged(Sensor sensor, int accuracy) {
   }
```

```
        final Handler mHandler = new Handler();
        final Runnable mUpdateResults = new Runnable() {
               public void run() {
                  updateGui();
               }
        };

        public void run() {
               while (true) {
                      if (refreshing) { mHandler.post(mUpdateResults); }
                      try {
                      Thread.sleep(200);
               }      catch (InterruptedException e) {
                      e.printStackTrace();
               }
               }
        }


  }
```

7.2 Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent" android:layout_height="fill_parent"
        android:orientation="vertical">
```

```
<RelativeLayout android:layout_width="fill_parent"
	android:layout_height="wrap_content" android:orientation="vertical">

	<TextView android:id="@+id/richtung" android:layout_width="wrap_content"
		android:layout_height="wrap_content" android:textSize="15dip" android:textColor="@color/green"
		android:text="@string/richtung" />
	<TextView android:id="@+id/richtungwert"
		android:layout_toRightOf="@id/richtung" android:paddingLeft="120dip"
		android:layout_width="wrap_content" android:layout_height="wrap_content"
		android:textSize="15dip" android:layout_gravity="right"
		android:maxLength="5"/>
	<TextView android:id="@+id/richtungwertS"
		android:layout_toRightOf="@id/richtungwert" android:paddingLeft="50dip"
		android:layout_width="wrap_content" android:layout_height="wrap_content"
		android:textSize="15dip" android:layout_gravity="right"
		android:maxLength="5"/>

	<TextView android:id="@+id/distanz" android:layout_width="wrap_content"
		android:layout_height="wrap_content" android:textSize="15dip" android:textColor="@color/green"
		android:text="@string/distanz" android:layout_below="@id/richtung" />
	<TextView android:id="@+id/distanzwert"
		android:layout_toRightOf="@id/richtung"
		android:layout_width="wrap_content" android:layout_height="wrap_content"
		android:textSize="15dip" android:layout_gravity="right" android:paddingLeft="120dip"
		android:layout_below="@id/richtung" android:maxLength="5" />
	<TextView android:id="@+id/distanzwertS"
		android:layout_toRightOf="@id/richtungwert"
```

```
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:textSize="15dip" android:layout_gravity="right" android:paddingLeft="50dip"
            android:layout_below="@id/richtung" android:maxLength="5"/>

        <TextView android:id="@+id/reldistanz" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:textSize="15dip" android:textColor="@color/green"
            android:text="@string/reldistanz" android:layout_below="@id/distanz" />
        <TextView android:id="@+id/reldistanzwert"
            android:layout_toRightOf="@id/richtung"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:textSize="15dip" android:layout_gravity="right" android:paddingLeft="120dip"
            android:layout_below="@id/distanz" android:maxLength="5" />
        <TextView android:id="@+id/reldistanzwertS"
            android:layout_toRightOf="@id/richtungwert"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:textSize="15dip" android:layout_gravity="right" android:paddingLeft="50dip"
            android:layout_below="@id/distanz" android:maxLength="5"/>
        <TextView android:id="@+id/infotext"
            android:layout_width="wrap_content" android:layout_below="@id/reldistanz"
        android:layout_height="wrap_content" android:layout_toRightOf="@id/richtung"
        android:layout_gravity="center" android:paddingLeft="120dip"
        android:text="@string/infotext"
        android:textColor="@color/gray"
        android:textSize="10dip" />
</RelativeLayout>


<TextView android:paddingBottom="6dip" android:paddingTop="6dip"
```

```
                        android:layout_marginTop="6dip" android:layout_marginBottom="6dip"
                        android:layout_height="2dip" android:layout_width="fill_parent"
                        android:background="@color/gray" />

<RelativeLayout android:layout_width="fill_parent"
      android:layout_height="wrap_content" android:orientation="vertical">

      <TextView android:id="@+id/ueberschrift2" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:textSize="10dip" android:textColor="@color/green"
            android:text="@string/ueberschrift2" />

      <TextView android:id="@+id/lina" android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:textSize="10dip" android:textColor="@color/yellow"
            android:text="@string/lina" android:layout_below="@id/ueberschrift2" />
      <TextView android:id="@+id/linawert1"
            android:layout_toRightOf="@id/lina" android:paddingLeft="30dip" android:layout_below="@id/ueberschrift2"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:textSize="10dip" android:layout_gravity="right" android:maxLength="5" />
      <TextView android:id="@+id/linawert2"
            android:layout_toRightOf="@id/linawert1" android:paddingLeft="30dip" android:layout_below="@id/ueberschrift2"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:textSize="10dip" android:layout_gravity="right" android:maxLength="5"/>
      <TextView android:id="@+id/linawert3"
            android:layout_toRightOf="@id/linawert2" android:paddingLeft="30dip" android:layout_below="@id/ueberschrift2"
            android:layout_width="wrap_content" android:layout_height="wrap_content"
            android:textSize="10dip" android:layout_gravity="right" android:maxLength="5"/>

      <TextView android:id="@+id/rotv" android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content" android:textSize="10dip" android:textColor="@color/yellow"
        android:text="@string/rotv"
        android:layout_below="@id/lina" />
<TextView android:id="@+id/rotvwert1"
        android:layout_toRightOf="@id/lina" android:paddingLeft="30dip"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/linawert1"
        android:maxLength="5"/>
<TextView android:id="@+id/rotvwert2"
        android:layout_toRightOf="@id/linawert1" android:paddingLeft="30dip"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/linawert2"
        android:maxLength="5"/>
<TextView android:id="@+id/rotvwert3"
        android:layout_toRightOf="@id/linawert2" android:paddingLeft="30dip"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/linawert3"
        android:maxLength="5"/>

<TextView android:id="@+id/gyro" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:textSize="10dip" android:textColor="@color/yellow"
        android:text="@string/gyro"
        android:layout_below="@id/rotv" />
<TextView android:id="@+id/gyrowert1"
        android:layout_toRightOf="@id/lina" android:paddingLeft="30dip"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/rotvwert1"
        android:maxLength="5"/>
```

```
<TextView android:id="@+id/gyrowert2"
	android:layout_toRightOf="@id/linawert1" android:paddingLeft="30dip"
	android:layout_width="wrap_content" android:layout_height="wrap_content"
	android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/rotvwert2"
	android:maxLength="5"/>
<TextView android:id="@+id/gyrowert3"
	android:layout_toRightOf="@id/linawert2" android:paddingLeft="30dip"
	android:layout_width="wrap_content" android:layout_height="wrap_content"
	android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/rotvwert3"
	android:maxLength="5"/>

<TextView android:id="@+id/zaehler" android:layout_width="wrap_content"
	android:layout_height="wrap_content" android:textSize="10dip" android:textColor="@color/yellow"
	android:text="@string/zaehler"
	android:layout_below="@id/gyro" />
<TextView android:id="@+id/zaehlerwert1"
	android:layout_toRightOf="@id/lina" android:paddingLeft="30dip"
	android:layout_width="wrap_content" android:layout_height="wrap_content"
	android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/gyro"
	android:maxLength="5"/>
<TextView android:id="@+id/zaehlerwert2"
	android:layout_toRightOf="@id/linawert1" android:paddingLeft="30dip"
	android:layout_width="wrap_content" android:layout_height="wrap_content"
	android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/gyro"
	android:maxLength="5"/>

<TextView android:id="@+id/magf" android:layout_width="wrap_content"
	android:layout_height="wrap_content" android:textSize="10dip" android:textColor="@color/yellow"
```

```
        android:text="@string/magf"
        android:layout_below="@id/zaehler" />
    <TextView android:id="@+id/magfwert1"
        android:layout_toRightOf="@id/lina" android:paddingLeft="30dip"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/zaehler"
        android:maxLength="5"/>
    <TextView android:id="@+id/magfwert2"
        android:layout_toRightOf="@id/linawert1" android:paddingLeft="30dip"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/zaehler"
        android:maxLength="5"/>
    <TextView android:id="@+id/magfwert3"
        android:layout_toRightOf="@id/linawert2" android:paddingLeft="30dip"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:textSize="10dip" android:layout_gravity="right" android:layout_below="@id/zaehler"
        android:maxLength="5"/>

    </RelativeLayout>

    <TextView android:paddingBottom="6dip" android:paddingTop="6dip"
                        android:layout_marginTop="6dip" android:layout_marginBottom="6dip"
                        android:layout_height="2dip" android:layout_width="fill_parent"
                        android:background="@color/gray" />

    <view class="at.ronkdev.gmh.RichtungView"
        android:id="@+id/compassview" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
                    android:background="@color/green"
                    android:layout_gravity="center_horizontal"
                    />

<!--          <TextView -->
<!--              android:id="@+id/kalib" -->
<!--              android:layout_width="fill_parent" -->
<!--              android:layout_height="wrap_content" -->
<!--              android:layout_gravity="center_horizontal" -->
<!--              android:textSize="5dip" /> -->



</LinearLayout>
```

7.3 RichtungView.java

```
package at.ronkdev.gmh;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.AttributeSet;
import android.util.Log;
import android.view.View;
```

```
public class RichtungView extends View {

    private Paint zeichenfarbe = new Paint();
    private Paint kreis = new Paint();

    private float winkel1=0;
    private float winkel2=0;
    private float winkel3=0;

    public RichtungView(Context context) {
        super(context);
        init();
    }

    public RichtungView(Context context, AttributeSet attrs) {
        super(context, attrs);
        init();
    }

    public RichtungView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs,defStyle);
        init();
    }

    private void init() {
        zeichenfarbe.setAntiAlias(true);
        zeichenfarbe.setColor(Color.WHITE);
```

```
    zeichenfarbe.setStyle(Paint.Style.FILL);
    kreis.setAntiAlias(true);
    kreis.setColor(Color.GRAY);
    kreis.setStrokeWidth(10);
    kreis.setStyle(Paint.Style.STROKE);
}


public void setWinkel1(float winkel) {
         this.winkel1 = winkel;
         invalidate();
   }

public void setWinkel2(float winkel) {
         this.winkel2 = winkel;
         invalidate();
   }

public void setWinkel3(float winkel) {
         this.winkel3 = winkel;
         invalidate();
   }


@Override
protected void onDraw(Canvas canvas) {

    canvas.drawColor(Color.BLACK);
```

```
        int breite = 475; //canvas.getWidth();
        int hoehe = 415; //canvas.getHeight();
        int laenge = 400;
        Path pfad = new Path();
        pfad.moveTo(0, -laenge/2);
        pfad.lineTo(laenge/20, laenge/2);
        pfad.lineTo(-laenge/20, laenge/2);
        pfad.close();

              canvas.translate(breite/2, hoehe/2);
              canvas.rotate(winkel1);
              zeichenfarbe.setColor(Color.WHITE);
              canvas.drawPath(pfad, zeichenfarbe);
              canvas.rotate(-winkel1);
              zeichenfarbe.setColor(Color.YELLOW);
              canvas.rotate(winkel2);
              canvas.drawPath(pfad, zeichenfarbe);
              canvas.rotate(-winkel2);
              zeichenfarbe.setColor(Color.GREEN);
              canvas.rotate(winkel3);
              canvas.drawPath(pfad, zeichenfarbe);
              canvas.rotate(-winkel3);

              canvas.drawCircle(0,0, 200, kreis);
              canvas.drawCircle(0,0, 12, kreis);

    }
```

```
}
```

7.4 MovingAverage.java

```
package at.ronkdev.gmh;

public class MovingAverage {

		private float circularBuffer[] = new float [2];
		private float avg = 0;
		private int circularIndex = 0;
		private int count = 0;


		public float getValue()
		{
			return avg;
		}

		public void pushValue(float x)
		{
			if (count++ ==0)
			{
```

```
                primeBuffer(x);
        }
        float lastValue=circularBuffer[circularIndex];
        avg = avg + (x - lastValue) / circularBuffer.length;
        circularBuffer[circularIndex] = x;
        circularIndex = nextIndex(circularIndex);

}

public long getCount()
{
        return count;
}

private void primeBuffer(float val)
{
        for (int i = 0; i < circularBuffer.length; ++i)
        {
                circularBuffer[i] = val;
        }
        avg=val;
}

private int nextIndex(int curIndex)
{
        if (curIndex + 1<= circularBuffer.length)
        {
                return 0;
```

```
                }
                return curIndex + 1;
            }
        }
```

8 Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Wien, den 8. Oktober 2012

Ronald Kränzl